INRIA

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 605

CHAÎNAGE EFFICACE DE CONTOUR

Gérard GIRAUDON

Février 1987

Chaînage Efficace de Contour

An Efficient Edge Chaining Algorithm

Gérard Giraudon

INRIA
Institut National de Recherche en Informatique et Automatique
Route des Lucioles
Sophia-Antipolis
06560 Valbonne FRANCE

Mots clés : Vision par ordinateur, Chaînage de contour, Extraction d'attributs, Automate à états finis.

RESUME :

Nous présentons dans ce papier une méthode efficace pour réaliser la construction des chaînes à partir d'une image de type contour. Notre algorithme se compose de trois phases simples et distinctes : une phase de création de chaînes en parallèle en une seule passe sur l'image qui s'exprime comme un automate à états finis, une phase de fusion, une phase d'élimination suivant un critère de longueur. Le résultat obtenu est une ensemble minimal de chaînes avec tous les liens de filiations conservés. La seule contrainte demandée est que les contours soient d'épaisseur un presque partout. Des résultats expérimentaux sont donnés en insistant largement sur la qualité de l'information résultante et sur la rapidité d'exécution du programme. La faisabilité temps réel est démontrée.

ABSTRACT :

In this paper, a new efficient chain-generating algorithm for the edge map data is presented. Our algorithm, performed in single pass over the image, is composed by three simple and distinct steps : a creation step realized by a finite states automata, a fusion step, a elimination step by chain length criterius. The result is a minimal set of chains. Every chain is a set of pixel coordinates, with the junction information about the intersection with the other chains. The edge map of one pixel thickness is the only constraint required. The quality of result information and the very low consuming CPU time are demonstrated by means of illustrations on real scenes egde maps. The real time feasibility is shown.

TABLES DES MATIERES

1. INTRODUCTION

Le but final de l'analyse de scène en vision par ordinateur, est de développer une représentation symbolique puis sémantique de l'information contenue dans une représentation numérique 2-D ou 3-D. Pour cela à partir de l'image numérique représentée la plupart du temps sous forme matricielle, la chaîne de traitements mise en jeu, va consister à élaborer des informations de plus en plus symboliques en vue de reconnaissance ou mieux encore en vue d'interprétation.

Ce changement d'état s'accompagne bien sur d'un changement de structure de données. Une des étapes les plus courantes dans cette élaboration consiste à extraire ce que l'on appelle les contours de l'image. Ces contours sont sensés représenter les frontières entre les différents objets qui composent la scène. L'extraction de contour se décompose en deux étapes :

- la détection de pixels contour
- le chaînage de ces pixels.

La détection de pixel contour s'effectue la plupart de temps avec des opérateurs de dérivation (dérivée première ou seconde). Ils fournissent une image binaire (au sens matriciel) de pixels contour d'épaisseur 1. Vient ensuite l'étape de chaînage où l'on va relier tous les points connexes entre eux, pour former une liste de contours chainés. C'est donc à cette étape qu'à lieu le changement de structure de données. C'est avec ces listes comme données d'entrée que d'autres processus vont commencer à travailler toujours dans le sens d'une généralisation, d'une élévation symbolique de l'information (chaînes -> segments -> attributs de forme -> objet).

L'étape de chaînage est à première vue juste un passage transitoire qui doit être aisé de réaliser. En fait il s'agit d'une étape délicate dans la chaîne de traitement, et cela pour trois raisons :

- La première est une question de temps calcul. En effet si nous, humains, n'avons pas de gros problème pour effectuer un chainage entre primitives, il n'en est pas de même avec un ordinateur. Il faut apporter le plus grand soin à sa réalisation sous peine de voir les temps calculs devenir très rapidement prohibitifs, pour une étape qui ne paraissait que transitoire.
- La deuxième raison est beaucoup plus fondamentale. En effet, toujours pour nous humains, il n'y a pas grande différence visuelle entre une image de contours et une image de chaîne de contours. Or pour un ordinateur la différence est énorme. En effet, dans une image de contour, nous sommes encore au stade matriciel de la représentation de l'information. Il n'y aucun lien, aucune information contextuelle entre les pixels, autre que que leur position respective dans la matrice. Dans la représentation par chaine de contour, nous avons matérialisé les liens entre pixels,

nous avons unifié l'information de contexte entre pixels connexes sous forme d'une entité plus générale que le pixel, la chaîne ou encore sous sa forme informatique la liste. Mais en faisant cela, il faut faire très attention à ne pas avoir perdu d'information potentielle qui pouvait se trouver dans la représentation matricielle. L'exemple le plus frappant est sans doute celui de la gestion des jonctions entre chaines. Si on ne gère pas cette information (chaine X a un lien avec la chaine Y et la chaine Z), on peut alors affirmer que l'on perdu de l'information par rapport la représentation matricielle. On ne mesure que ce que l'instrument de mesure peut mesurer, le reste est détruit, il ne faut pas l'oublier.

- La troisième raison est liée à la question : à quoi et comment doit servir l'information extraite. En effet, les données en vision par ordinateur doivent souvent comparées entre elles ou par rapport à un contexte, modifiées partiellement ou totalement, globalisée soit numériquement ou symboliquement. Il faut donc bien structurer l'information pour que les manipulations des données à effectuer par la suite par un quelconque processus informatique puisent être aisées à la fois pour un gain de temps et surtout pour une plus grande souplesse de couplage, de recoupement, de combinaison de l'information qui doit être à nous yeux la grande potentialité pour les systèmes intelligents à venir en vision par ordinateur.

Le problème de la connectivité et plus généralement du 'Computational Geometry' a été un des premiers thème abordé en Vision [Min 69], [Ros 70]. Différents auteurs [Cha 81] [Kul 78] [Ros 78] [Fre 74] [Sob 78] ont déjà présenté des algorithmes pour essayé de résoudre le problème du chaînage. Beaucoup se placent dans un contexte de chainage de frontière de tâches binaires [Kul 78], [Sob 78] ou encore ne traitent pas le problème de jonctions dans le cadre d'image de contour [Ros 78], ou encore imposent deux passes sur l'image [Pav 78]. Un des travaux les plus intéressants est celui proposé par Chakravarty [Cha 81], qui se place dans le contexte d'image de contour avec traitement des jonctions. A titre de remarque, nous pouvons constater que le problème du chainage a été étudié très tôt, mais qu'à notre avis il n'y avait pas encore de solution élégante, générale et rapide du problème du chainage sur image de contour.

Nous proposons dans ce rapport une nouvelle méthode de chaînage rapide et générale sur des données matricielle de type contour, en essayant de tenir compte des problèmes évoqués ci-dessus, que l'on a synthétisé sous les contraintes suivantes :

- chaînage rapide en une seule lecture de l'image.
- gestion efficace des liens de parenté
- possibilité de stockage étendu pour divers types d'information (niveau de gris pixel, amplitude gradient , etc..)

- possibilité d'élimination de chaînes : nous avons choisi dans un premier temps un critère de longueur
- minimisation du nombre de chaînes
- possibilité de travailler avec des stockages d'image réduit par appel itératif de l'image depuis le support externe.
- faisabilité temps réel envisageable pour un coût réduit.

Pour réaliser cela, nous avons repris l'idée de décomposition d'un pavé 3x3 centré autour d'un pixel, en passé, présent, futur, idée déjà utilisé pour le coloriage de tâches [Bal 82], ou sur les problèmes de restauration d'image (modèle Arma), ou encore par [Cha 81]. L'avantage de la méthode présenté ici par rapport aux précédentes et en particulier par rapport à [Cha 81] est de développer d'une part des concepts plus généraux liés à la notion de chaînes et d'autre part des concepts beaucoup plus simples, donc plus rapide, dans leur réalisation, entre autre sur les points suivants :

- décomposition d'un problème complexe en trois étapes simples distinctes et indépendantes : création de chaînes en parallèle, en une passe, fusion de chaîne pour réaliser la minimisation du nombre de chaînes, élimination de chaînes suivant un critère. La puissance de notre méthode vient alors de la combinaison de ces trois processus.
- réduction du nombre de tests pour la gestion des jonctions
- ajout immédiat du pixel dans la chaîne par simple manipulation de pointeur au lieu d'une gestion lourde de lien symbolique
- ne nécessite pas d'orientation prédéfinie des chaînes de type "sens des aiguilles d'une montre" (symétrie des rôles joués par la tête et la queue de la chaîne)
- possibilité de détection de sous ensembles de chaînes remarquables de manière simple.

Après cette introduction, nous présentons plus en détail, au chapitre II, le problème général du chaînage et les idées de base ainsi que notre méthodologie pour le résoudre par le découpage en trois sous problèmes distincts, le chaînage, la fusion et l'élimination. Nous présentons ensuite les différentes étapes de traitement. Tout d'abord au chapitre III, nous montrons la mise en oeuvre de la partie chaînage proprement dite et sa formulation sous forme d'automate à états finis. Au chapitre IV, nous présentons l'algorithme de fusion et la manière de résoudre les problèmes de modification des filiations. Au chapitre V, nous abordons le problème de l'élimination, et les problèmes pratiques de mise en oeuvre. Au chapitre VI, nous proposons la méthode générale de chaînage et montrons que la solution fusion-élimination est optimale au sens de la conservation des chaînes pouvant répondre au critère. Enfin au chapitre VII, nous illustrons les possibilités de la méthode sur des images réelles et

présentons les performances en temps calculs obtenues pour une solution implémentée en langage C. Pour répondre à la question de la faisabilité temps réel, nous présentons la complexité de l'algorithme et une implémentation parallèle réaliste avec automate. Pour finir, nous essayons de faire la synthèse de ce travail, et nous parlons des conséquences et perspectives induites.

2. LE PROBLEME DU CHAINAGE

2.1. CONTRAINTES ET OBJECTIFS

On désire réaliser un algorithme de construction de chaînes de contour efficace aussi bien au niveau rapidité d'exécution (construction temps réel sur un signal de type vidéo) qu'au niveau qualité de l'information construite (pas de perte d'information pertinente)

La nature même de l'acquisition d'un signal vidéo impose un traitement local ligne à ligne évidemment en un seul passage sur l'image. Il s'agit donc de créer de gérer les chaînes en parallèle en balayant l'image de manière séquentielle en haut en bas et de gauche à droite.

Le rythme d'acquisition vidéo, de l'ordre de 100 ns par pixel, impose un traitement relativement sommaire. Or le principe du chainage est avant tout une mise à jour cas par cas, des chaînes (accès aléatoire à de la mémoire) et non pas un processus de type calcul flottant à effectuer sur tous les pixels. Le problème est donc complexe : comment effectuer, toutes les 100 ns un traitement mettant en jeu plusieurs accès aléatoires sur de la mémoire RAM (entre 100 et 200 ns de temps d'accès au mieux).

On suppose que l'image à chaîner est une image de type contour obtenue par un algorithme quelconque. On imposera seulement que les contours soient d'épaisseur un presque partout. En effet de nombreux détecteurs de contour qui devraient donner des contours d'épaisseur un commettent quelquefois des erreurs de type détection d'un pavé 2x2. Plutôt que de réaliser une opération spécifique (type amincissement) pour ces quelques erreurs, nous préférons réaliser un chainage qui soit capable de les gérer.

Au niveau qualité de l'information, nous désirons gérer les jonctions, c'est à dire détecter les points triples ou quadruples et mémoriser les liens qui unissent les chaînes connexes à ce type de pixel. Cette partie doit être parfaitement réalisée. Gérer une jonction demande d'abord de la détecter. La détection demande à accéder rapidement au voisinage du pixel. Le voisinage du pixel est constitué des pixels connexes appartenant à la ligne au dessus et à la ligne au dessous. Compte tenu du signal vidéo cela signifie que l'on est obligé de travailler avec trois lignes image, la ligne du dessus étant déjà traitée. Cela signifie aussi que l'on a une ligne de retard par rapport au signal vidéo. Dans cet environnement là, la difficulté va consister à minimiser les fausses jonctions dans la détection des jonctions potentielles.
L'information sera du type : la chaîne X a un lien avec la chaîne Y et la chaîne Z.

La qualité de l'information se mesure aussi par l'absence de bruit ou d'information non pertinente. Pour cela nous désirons pouvoir éliminer des chaînes considérées comme du bruit. Deux notions sont utilisées pour définir un bruit. D'abord un critère de longueur de la chaîne (on veut avoir la possibilité d'éliminer les plus petites), deuxièmement un critère de non pertinence, il faut en plus que la chaîne ne présente

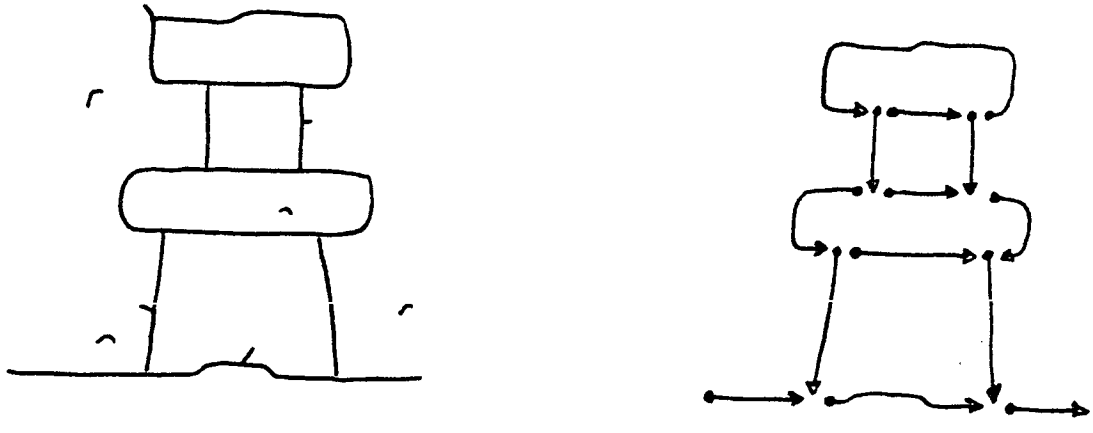
pas de liens dont la perte correspondrait à une perte d'information.

Exemple : La longueur de la chaîne X est plus petite que le seuil
et la chaîne X a des liens au plus sur une extremité,
alors la chaîne X peut être éliminée.

Pour des problèmes évidents de représentation de l'information, nous désirons minimiser le nombre de chaînes donc avoir les chaînes les plus longues possibles.

Exemple : après élimination de la chaîne X,
les chaînes Y et Z peuvent fusionner.

Nous illustrons sur un petit exemple la représentation finale du chainage à laquelle nous voulons arriver :



l'image bi_dimensionnelle

les chaînes obtenues

- figure No 1 -

Chacune des chaînes est une liste de pixels connexes et le symbole <-> marque un lien de filiation.

En résumé, les objectifs que nous voulons atteindre avec la réalisation de cette méthode se situent sur deux plans.

Le premier concerne la qualité d'implémentation : temps CPU - mémoire requise

- chaînage rapide en une seule lecture de l'image et réalisation temps réel envisageable pour un coût réduit.
- minimisation du nombre de chaînes nécessaire pour la représentation de l'information et possibilité de travailler avec des stockages d'image réduit par appel itératif de l'image depuis le support externe (nécessaire pour une implémentation temps réel, intéressant pour une version logicielle).

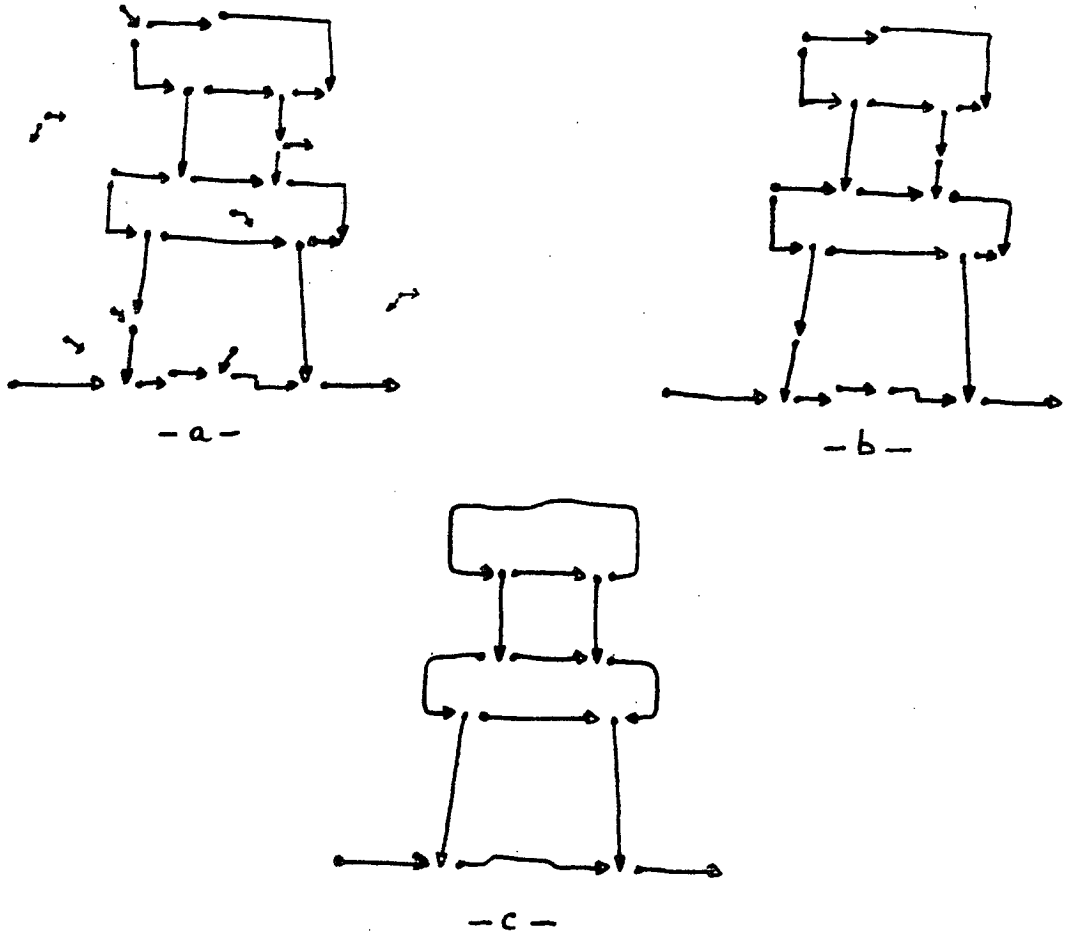
Le deuxième concerne la qualité de l'information extraite : l'information sur une chaîne, l'information sur les liens entre chaînes.

- possibilité de stockage étendu pour divers types d'information (niveau de gris pixel, amplitude gradient , etc..)
- gestion efficace des liens de parenté
- possibilité d'élimination de chaînes : nous choisissons dans un premier temps un critère de longueur

2.2. IDEES DE BASE

Pour obtenir une telle représentation par traitement de l'image ligne par ligne avec les contraintes exposées ci-dessus, nous avons utilisé les idées de base suivantes:

- Connexité : Principe du coloriage de tache [Bal 82], balayage de l'image par un pavé 3x3 en 8-connexité
- Gestion des chaînes : Utilisation d'une mémoire recouvrant deux lignes image qui regroupe toutes les informations sur les pixels déjà traités.
  Utilisation d'une structure de l'information de type liste avec pointeur.
- Détection des jonctions potentielles. Lorsque celles-ci sont trouvées, fermeture des chaînes ouvertes, ouverture d'une nouvelle chaîne, création des liens de filiation.
- Complexité du problème : décomposition en trois étapes distinctes , solution qui minimise la complexité algorithmique et qui est capable de traiter toutes les configurations possibles :
  - étape N0 1 : Le création - traitement effectué en un seul balayage image
    Sur l'exemple précédent cela se traduit par la figure 2-a
  - étape N0 2 : l'élimination - traitement effectué sur les listes
    Sur l'exemple précédent cela se traduit par la figure 2-b
  - étape N0 3 : la fusion - traitement effectué sur les listes
    Sur l'exemple précédent cela se traduit par la figure 2-c



- figure No 2 -

2.3. DEFINITIONS

Définition 1 :

On suppose que l'image en entrée est représentée par une matrice M de QxP pixels. Chaque **pixel** est caractérisé par

- ses coordonnées dans l'image.
- son niveau de gris.

On dit qu'un pixel de coordonnées (i,j) est **significatif** si son niveau de gris est différent de 0, c'est à dire si M(i, j) # 0. Les pixels qui sont au bord de l'image ne sont pas significatifs. On suppose que les pixels significatifs représente un contour d'épaisseur 1, presque partout.

Définition 2 :

Nous travaillerons en **8-connexité** dans laquelle nous distinguerons 2 types de voisins, équivalent à deux types de 4-connexité. Il y a un type que nous appellerons les **d-voisins** et l'autre type **i-voisins** définis ainsi :

i	d	i
d	.	d
i	d	i

d = d_voisins
i = i_voisins

Définition 3 :

On décompose le pavé 3x3 centré autour du pixel courant comme étant composé d'un passé , d'un futur et d'un présent

Pa	Pa	Pa
Pa	C	Fu
Fu	Fu	Fu

C le présent
Pa le passé
Fu le futur.

Définition 4 :

Classiquement, on définit une détection de jonction potentielle pour le pixel courant si

- il existe deux pixels passés et au moins un pixel futur
- ou il existe deux pixels futurs et au moins un pixel passé.

Exemples de jonctions potentielles.

x		x
	x	
x		

x	x	
	x	x
x		

.	.	.
x	x	x
	x	

Définition 5 :

On distingue dans toute chaîne deux extrémités :

- la **tête** : le premier maillon incorporé dans l'ordre de création dans la chaîne.
- la **queue** : le dernier maillon incorporé dans l'ordre de création dans la chaîne.

Définition 6 :

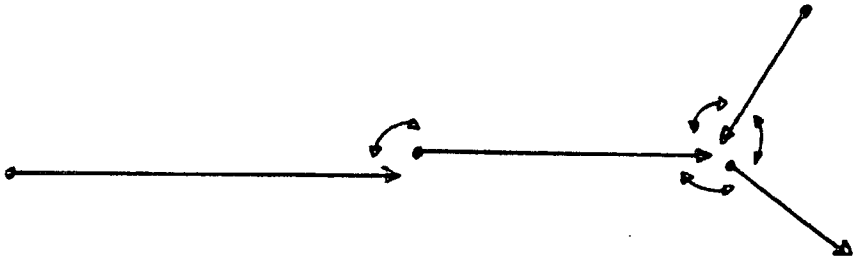
Pour chacune des extrémités, on établit **un lien de filiation.**
Un lien de filiation est une information qui permet de de savoir

- quelles sont les autres chaînes connexes
- et de quel type, tête ou queue, est l'extrémité de l'autre chaîne connexe

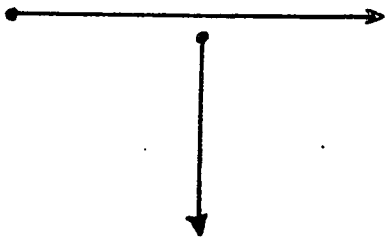
Nous n'imposons aucune contrainte d'impossibilité de jonction de type queue sur queue ou tête sur tête, contrairement à [Cha 81]. Il n'y a donc pas de direction privilégié pour une chaîne.

Exemple schématique : (<-> signifie que les chaînes ont un lien de filiation)

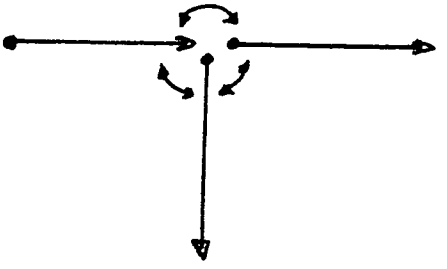
Ce qu'il est normal d'obtenir :



Ce qui ne doit pas se produire :



Ce qu'on aurait dû obtenir :



Définition 7 :

Nous définissons la structure "chaîne" comme étant composée de deux entités :

* un en-tête composé de :
- un numéro : c'est l'identificateur de la chaîne.
- une tête : un pointeur sur le premier pixel à avoir été inclus dans la chaîne.
- une queue : un pointeur sur le dernier pixel à avoir été inclus dans la chaîne.
- un lien de filiation sur tête : un pointeur sur la liste des numéros de chaînes associés à la tête.
- un lien de filiation sur queue : un pointeur sur la liste des numéros de chaînes associés à la queue.
- des capacités de stockage d'information générale sur la chaîne (nombre d'élément, valeurs moyennes, etc...)

* les maillons :
- un maillon est composé a priori par les coordonnées du pixel et son niveau de gris. On peut lui adjoindre tout type d'information supplémentaire.

3. CREATION ET GESTION DES CHAINES.

3.1. ETAPES IMPORTANTES DE L'ALGORITHME

On désire effectuer le chaînage de tous les pixels de l'image en une seule passe sur l'image et en ne perdant aucune information sur les jonctions rencontrées. Pour cela on balaye toute l'image, ligne par ligne, en déplaçant un pavé 3x3 et en ne traitant que les points significatifs. Au cours de ce parcours, les différentes chaînes sont créées et gérées en parallèle. Nous décrivons ci-dessous les différentes actions que l'on est succeptible d'effectuer pour chaque pixel courant et significatif, en fonction de son contexte passé et futur :

1/ On ouvre une nouvelle chaîne.

2/ On y inclut le premier point, ce point sera la tête de la chaîne et le restera dans toute la phase de chainage.

3/ On ajoute un point dans la chaîne. Ce point est appelé la queue de la chaîne. Cette étape peut se répéter autant de fois que nécessaire, étant bien entendu que c'est toujours le dernier point inclus dans la chaîne qui joue le rôle de queue.

4/ On ferme la chaîne. A partir de ce moment le dernier point inclus dans la chaîne est considéré comme la queue définitive, et le restera dans toute la phase de chainage. On s'interdit donc d'y ajouter de nouveaux points.

5/ On établit les liens de filiation en tête ou en queue.

6/ On teste l'existence de jonctions potentielles. Lorsque une jonction potentielle est détectée, on inclut le pixel courant dans une des chaînes connexes à ce pixel, et on ferme toutes les chaînes ouvertes connexes à ce pixel.

Nous décrivons maintenant en détail les grandes phases de l'algorithme : le traitement du pixel courant, le traitement de la filiation, la détection des jonctions.

3.1.1. LE TRAITEMENT DU PIXEL COURANT

Le traitement à effectuer pour un pixel significatif est fonction de son contexte c'est à dire de l'information contenue dans son voisinage. Son contexte est symboliquement divisé en deux entités temporelles.

- Information du passé = Informations sur les voisins déjà traités.
- Information du futur = Informations sur les voisins non encore traités.

Information du passé.

On définit trois classes possibles pour les pixels du passé :

1/ Pixel de chaîne ouverte
C'est le dernier pixel inclus dans une chaîne dans laquelle on s'autorise à ajouter des pixels.

2/ Pixel de chaîne fermée.
C'est le dernier pixel inclus dans une chaîne dans laquelle on s'interdit d'ajouter de nouveaux pixels.

3/ Autres pixels.
C'est un pixel n'appartenant à aucune des catégories précédentes. Se retrouvent dans cette classe tous les pixels non terminaux des chaînes ainsi que les pixels de tête.

A partir de ces 3 classes on peut associer à chaque pixel courant deux valeurs :

- Le nombre de chaînes ouvertes = **NCO** , c'est le nombre de pixels du passé classés dans la catégorie 1.
- Le nombre de chaînes fermées = **NCF** , c'est le nombre de pixels du passé classés dans la catégorie 2.

Information du futur.

Afin de détecter les jonctions potentielles, on ne considère que les informations relatives aux pixels du futur. Dans un premier temps elles se réduisent à un nombre :

- le nombre de pixels du futur = **NPF**

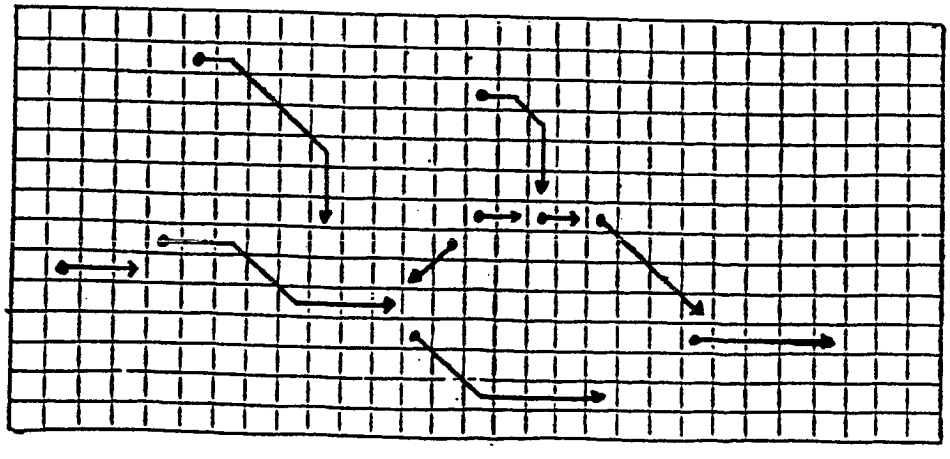
A partir des informations sur le passé et le futur du pixel, information condensé dans les valeurs NCO, NCF, NPF, on peut traiter le pixel courant suivant l'algorithme exposé ci-dessous. On ne tient pas compte ici du problème de la filiation.

L'algorithme :

```
                CHAINAGE( pixel, NCO, NCF, NPF )

case NCO
  NCO = 0
            | ouvrir une nouvelle chaîne i
            | inclure le pixel courant dans la chaîne i
            | ce pixel devient la tête de chaîne

  NCO = 1
            | inclure le pixel courant dans la chaîne ouverte
            | ce pixel est alors la queue de chaîne
            | si NCF >= 1 alors  fermer la chaîne ouverte
            |          sinon  si NPF >= 2 alors fermer la chaîne ouverte
            |                      finsi
            | finsi

 NCO = 2
            | inclure le pixel courant dans une des deux chaînes ouvertes
            | fermer les deux chaînes ouvertes

fincase
```

Remarque : l'algorithme implique que la propriéte NCO <= 2 reste toujours vraie.

Exemples de chaînes créées par cet algorithme :



3.1.2. LE PROBLEME DE LA FILIATION

La filiation consiste à établir pour chaque chaîne :

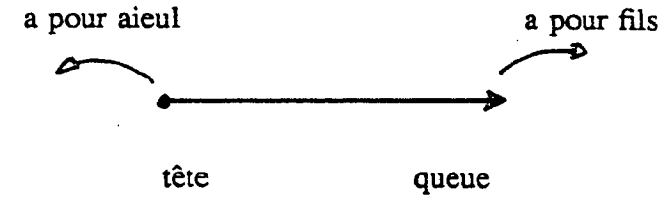
1/ la liste des numéros des chaînes contigue#s à la tête, ces chaînes seront appelées **aieuls.** Cette liste sera appelée **famille de tête.**

2/ La liste des numéros des chaînes contigue#s à la queue, ces chaînes seront appelées **fils.** Cette liste sera appelée **famille de queue.**

Cependant, le terme de **filiation** doit être compris dans la suite du texte comme un lien entre deux chaînes qui ont une extrémité dans un même voisinage et non pas comme un problème de descendance, ce qui permet de se passer de la notion d'orientation des chaînes, donc réalise un gain important de manipulation de données.

Ces deux listes sont directement accessibles par des pointeurs stockés dans l'en-tête de chaque chaîne. Ceci sera particulièrement intéressant pour la fusion et l'élimination.

Exemple de cas de mise en place de liens de filiation



Définition

On établit pour chaque jonction une double filiation :

c'est à dire que si i est lié à j (i a pour fils ou aieul j)
alors j est lié à i (j a pour fils ou aieul i).

En plus, si i est lié à j alors i est lié soit vers la tête
soit vers la queue de j.

On parlera par la suite d' **attribut de lien** pour désigner si un lien va vers une queue ou une tête.

Cette double filiation est mise en place à chaque fois qu'on ouvre ou qu'on ferme une chaîne lorsqu'on a détecté une jonction potentielle. Cette redondance d'information est nécessaire pour simplifier le problème de fusion (cf chapitre IV).

On pourrait penser dans un premier temps, que cette filiation doit être mise entre cette chaîne et les chaînes du passé qui lui sont contigue#s.
En fait, il y a deux cas où cette filiation ne doit pas être mise en place systématiquement :

1/ Il n'est inutile de mettre une filiation d'une chaîne sur elle même.

2/ Si parmi les chaînes du passé, il y en a deux qui sont connexes, alors forcément :

l'une est un i_voisin du pixel courant
et l'autre un d_voisin

Dans ce cas, on ne met un lien de filiation qu'avec le d_voisin.
Il est trivial de comprendre qu'alors par transitivité, tous les liens de filiation existent. Ce principe est mis en place pour simplifier l'étape de fusion.

Il est à noter que ces deux règles permettent de limiter à 4 le nombre de filiations à l'une quelconque des extrémités d'une chaîne . De plus, dans le cas particulier d'une chaîne de 1 pixel, il est préférable de mettre des liens de filiation uniquement au niveau de la famille en tête, afin d'éviter de faire un traitement particulier pour ces chaînes

Pour mettre en place cette filiation, les informations sur le passé doivent nous permettre, quand on ouvre ou ferme une chaîne, de connaître :

- les extrémités des chaînes voisines.
- de quel type elles sont : tête ou queue.

Pour connaître le type des extrémités voisines on associe à chaque extrémité de chaînes un type :

- 'T' : pour une tête de chaîne.
- 'Q' : pour une queue de chaîne.

La connaissance de ces informations nous permet alors de mettre en place sans ambiguité les liens de filiation. Cela est réalisé avec le souci de ne pas perdre d'information, et de simplifier la fusion.

3.1.3. DETECTION DES JONCTIONS POTENTIELLES

On cherche à obtenir un algorithme simple de construction des chaînes (pour des problèmes de rapidité) mais qui minimise le plus possible nombre de chaînes créées

(pour des problèmes de taille mémoire). Le problème de minimisation est ici directement lié à la capacité de détection des jonctions véritables (points triples). La difficulté du problème est donc de ne fermer une chaîne que lorsque c'est nécessaire, avec l'information du passé et une vision réduite du futur.

Une jonction est la congruence sur le pixel courant d'au moins deux chaînes. Sa détection doit servir à fermer les chaînes ouvertes et à marquer les liens de filiation. De plus, d'un point de vue algorithmique, on désire limiter le nombre de tests pour savoir si le pixel courant présente un environnement de jonction.

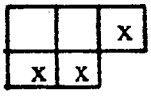
On peut rapidement s'apercevoir que la définition 4 n'est pas assez sélective et peut être améliorée En effet, il y a des cas dans lesquels l'algorithme initial ferme la chaîne par une mauvaise prédiction de la jonction et il y a des cas que cette définiton ne résoud pas.

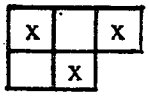
Nous proposons ci-dessous un nouveau critère de détection des jonctions qui répond à tous les cas particuliers et qui s'exprime simplement.

Définition 4bis

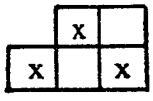
il y a jonction quand 2 chaînes qui conduisent à un même pixel courant ont des extrémités qui ne sont pas contigue#s dans l'image.

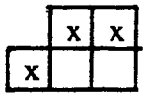
Dans le passé deux cas possibles :



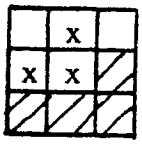


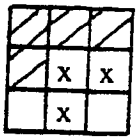
Dans le futur deux cas possibles :





Remarque : les cas ci-dessous sont inclus dans les cas précédents.





Essayons maintenant de voir s'il est possible de ramener les tests de jonction à un processus simple. Pour cela, numérotons les pixels contigus au pixel courant de la façon suivante :

T1	T2	T3
T4	X	P5
P6	P7	P8

avec Ti = vrai si Ti est l'extrémité d'une chaîne
= faux sinon

et Pi = vrai s'il existe un pixel dans la case i.
= faux sinon

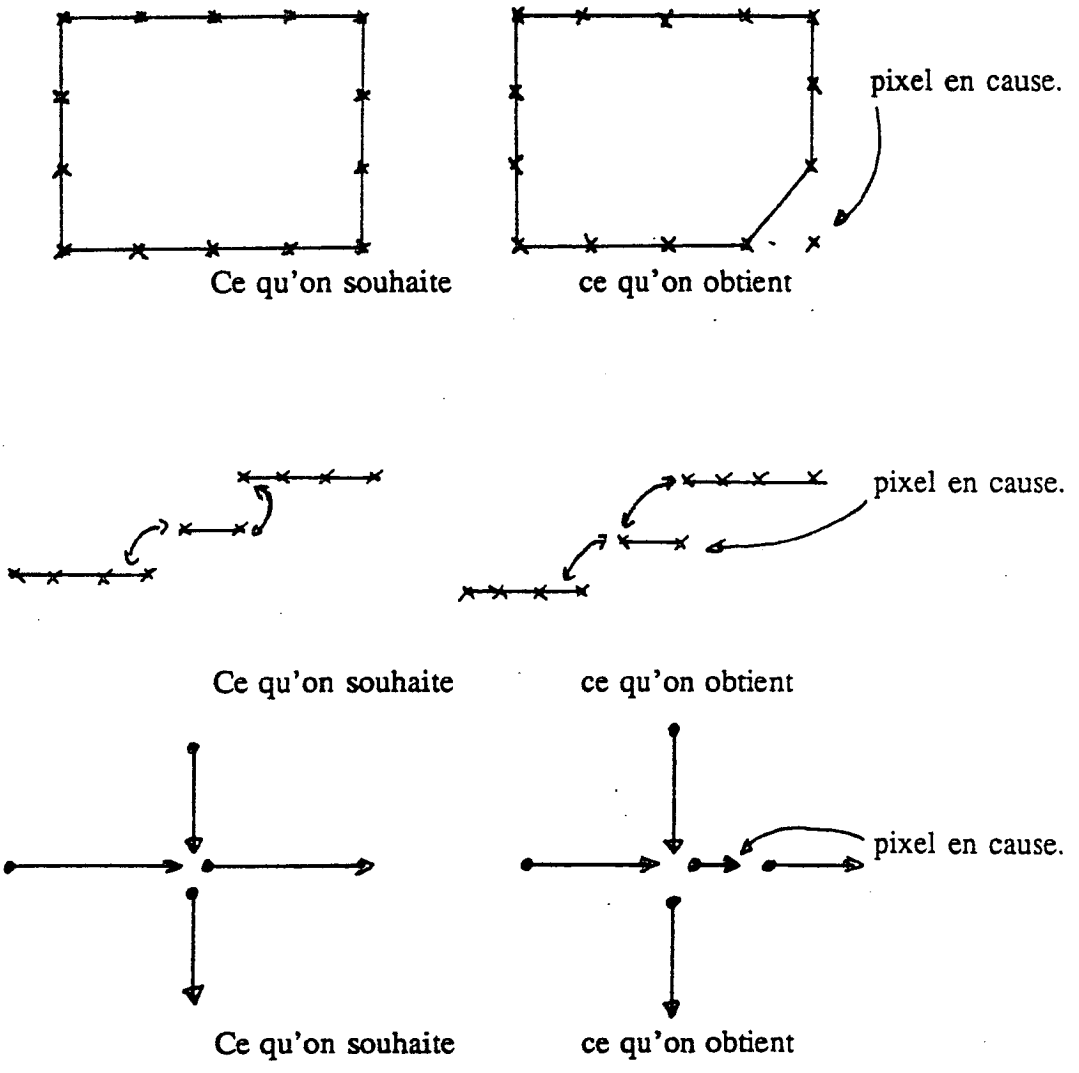
On obtient alors 4 tests logiques simples et généraux pour détecter une jonction potentielle. Ces tests s'énoncent ainsi :

Algorithme : une jonction est détectée pour le pixel courant si :
- pour le passé : (T1 et T3) ou (T3 et T4) vrai
- ou pour le futur : (P5 et P6) ou (P6 et P8) vrai

CAS PARTICULIER DES COINS CARRES.

Avec la définition donnée ci-dessus, on obtient des résultats non satisfaisants sur des configurations présentant des coins carrés : création de chaînes inutiles ou mauvaise mise en place des liens de filiation. Des exemples flagrants sont donnés sur la figure 3.

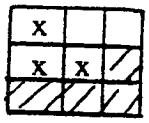
Exemple :

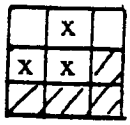


- figure 3 -

Pour répondre à ce type de problème, la solution consiste d'une part à l'incorporation du pixel en cause dans une des deux chaînes et d'autre part à mettre des liens sur les pixels les plus proches dans le cas de chaînes de longueur 1.

*) *Incorporation du pixel.*
On reconnait un coin quand on se trouve dans la configuration suivante :



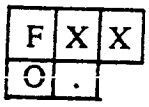


L'idée la plus simple, sans mettre en oeuvre une batterie de tests sur le futur trop importante, s'énonce ainsi :

> dans ce cas et dans ce cas là seulement, on s'autorise en inclure le pixel en cause dans une des deux chaînes, même lorsqu'elles sont fermées toutes les deux (fermeture réalisée le coup d'avant).

Les conséquences :

- On obtient moins de chaînes de 1 maillon
- On met moins de liens de filiation
- Tous les coins deviennent carrés
- On réduit à 1 cas particulier le traitement de chaînes de 1 maillon sur détection de jonction.
  Lorsqu'on se trouve dans le cas suivant :



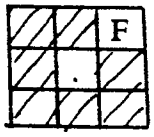
F = pixel de chaîne fermée.
O = pixel de chaîne ouverte de longueur 1.
X = pixel sans signification.

il est préférable de laisser la chaîne ouverte après y avoir incorporer le pixel courant .

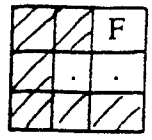
*) *Mise en place de liens de filiation plus directs*

Il faut établir les liaisons avec les chaînes les plus proches. En pratique cela signifie qu'il faut, dans certains cas, aller voir le pixel "suivant" du pixel courant pour savoir si on doit mettre ou non un lien de filiation. En fait, on teste si la prochaine configuration sera un coin carré.

on met un lien de filiation



On ne met pas de lien



Les conséquences :

- On est obligé de tester plus souvent le futur.
- Les filiations "respectent" mieux l'image .
- On met en place moins de liens de filiation.

**) Réduire le nombre de liens de filiations.*

On a déjà montré qu'il n'était pas nécéssaire de mettre en place une filiation avec tous les voisins (page 16)
Dans deux nouveaux cas on va encore réduire ces filiations.

1/ Il n'est pas nécessaire de mettre la liaison verticale. C'est fait par transitivité.

F	F	X
F	.	

2/ Quand les 2 chaînes fermées correspondent à la même chaîne, on ne met un lien qu'avec celle de gauche :

F	X	F
X	.	

Ou pas du tout quand :

F	X	F
F	.	

3.2. MISE EN OEUVRE EFFICACE DE L'ALGORITHME

Nous désirons que cet algorithme soit utilisé dans de nombreuses applications à vocation en temps réel. On cherche donc à obtenir un programme efficace tout en minimisant la place mémoire utilisée.

3.2.1. TRAITEMENT DE L'INFORMATION

3.2.1.1. L'information du passé

Pour chaque pixel, on doit être capable de connaître :
- les numéros associés aux extrêmités des chaînes voisines.
- le type de ces extrêmités (Tête ou Queue)
- le nombre de chaînes ouvertes qui conduisent à ce pixel.
- le nombre de chaînes fermées qui conduisent à ce pixel.

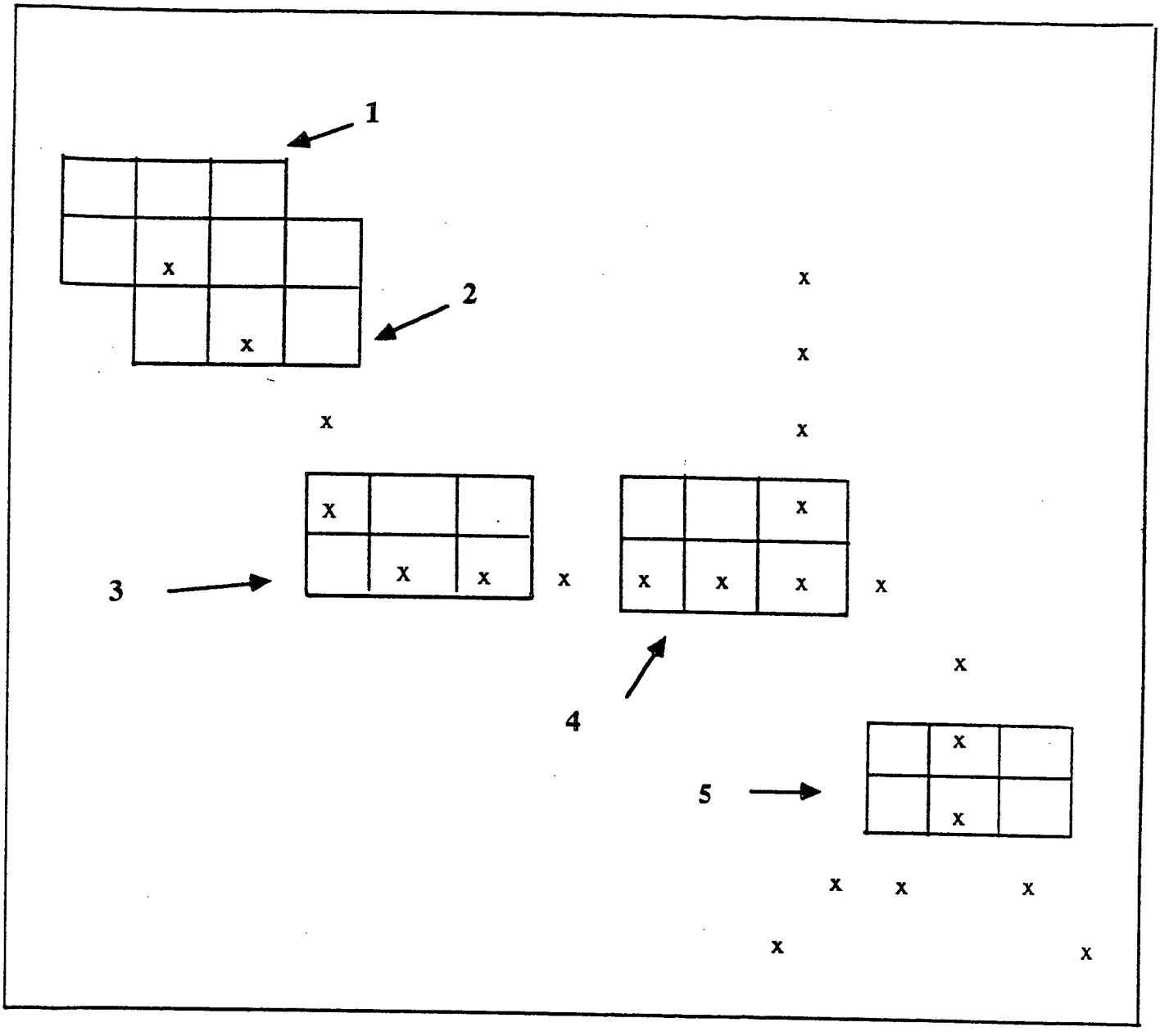
On définit alors une structure proche de la structure image, pour stocker au fur et à mesure de leur création et en parallèle les informations sur les chaînes. La structure la mieux adaptée nous semble être un tableau (ITAB) qui recouvre 2 lignes de l'image et de la même dimension en colonnes que l'image. En chaque pixel image du passé, on retrouve à la même position en colonne dans le tableau ITAB l'ensemble des informations y afférant.

Cette structure est un point clé dans la mise en oeuvre de la méthode, car c'est elle qui fait le lien entre les différentes lignes de l'image, en localisant spatialement et de manière rigoureuse, toutes les informations nécessaires à la poursuite des chaînes.

Au fur et à mesure qu'on traite les points de l'image, on met à jour les informations suivantes dans ITAB :
- le type des points traités.
- le numéro de la chaîne associée au point courant.

Le nombre de chaînes ouvertes (NCO) et le nombre de chaînes fermées (NCF) doit pouvoir être calculé à partir du type des points .

De plus, nous avons vu que le type des points devait également permettre de mettre en place la filiation.



- Figure n° 4 -

Pour résoudre ces deux problèmes, on est conduit à associer potentiellement à un pixel significatif de l'image l'un des 5 types suivant :

- 'D' : Début d'une chaîne ayant plus d'un élément.
- 'P' : Début d'une chaîne ayant un seul élément.
- 'O' : Queue d'une chaîne ouverte.
- 'F' : Queue d'une chaîne fermée.
- 'X' : Autre cas. (le point n'est pas ou n'est plus significatif).

Le nombre de chaînes ouvertes (et fermées) associé à un point se calcule maintenant en regardant le type des 4 points voisins du passé :

- NCF d'un pixel = (nombre de voisins ayant un type 'D')
  + (nombre de voisins ayant un type 'F')
- NCO d'un pixel = (nombre de voisins ayant un type 'O')
  + (nombre de voisins ayant un type 'P')

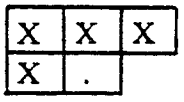
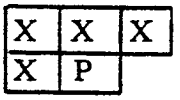
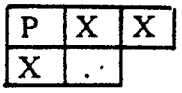
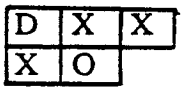
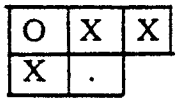
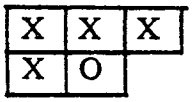
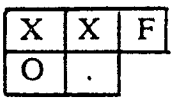
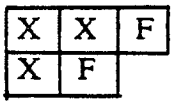
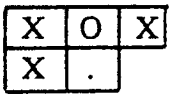
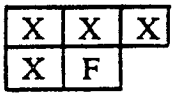
3.2.1.2. L'information du futur

Les informations du futur servent uniquement pour la détection d'une jonction potentielle. Soit i la ligne courante, il faut pouvoir accéder la ligne image i+1 et établir le test défini page 18 :

- (P5 et P6) ou (P6 et P8) vrai

3.2.1.3. Exemple de mise à jour des informations

Soit l'image représentée sur la figure 4 Les numéros dans l'image (de 1 à 5) correspondent à de configurations dont nous allons détailler le traitement.

	Avant le traitement	Après le traitement	Commentaires
1/			On ouvre une chaîne
2/			On inclut un point dans une chaîne ayant un seul maillon ('P' devient 'D').
3/			On inclut un point dans une chaîne ayant plus d'un maillon ('O' devient 'X').
4/			On ferme une chaîne à cause du passé.
5/			Ici c'est le futur de la chaîne qui oblige à une fermeture

3.2.2. AUTOMATE A ETATS FINIS

Nous avons vu au début de ce chapitre (page 14), que la suite des procédures à exécuter pour réaliser la création et la gestion des chaînes était entièrement déterminée par la connaissance des valeurs de NCO et NCF concernant les pixels du passé, et par un test sur la configuration future (page 18).

Pour la détermination de NCO et NCF, on considère que chaque pixel du passé peut être d'un des types suivants :

- O pour une chaîne ouverte. Ici le type 'P' est confondu avec 'O'
- F pour une chaîne fermée. Ici le type 'D' est confondu avec 'F'
- X pour les autres cas.

A partir de là, il n'y a qu'un nombre fini de possibilités de combinaison (O,F,X) pour le passé du pixel. En assimilant chaque combinaison à un état, nous ramenons le problème à la définition d'un automate à états finis [Sel 72].

3.2.2.1. Nombre d'états possibles

*) Nombre d'états où nco (nombre de chaînes ouvertes) = 0

$$2^4 = 16 \text{ cas}$$

Par exemple si l'on note (XXXF) l'état représentant la configuration suivante :

X	X	X
F	.	

On a alors les états suivants :

- Etat1 = XXXX
- Etat2 = XXXF , Etat3 = XXFX , Etat4 = XFXX , Etat5 = FXXX
- Etat6 = XXFF , Etat7 = XFXF , Etat8 = FXXF , Etat9 = FXFX, Etat10 = FFXX , Etat11 = XFFX
- Etat12 = XFFF , Etat13 = FXFF , Etat14 = FFXF , Etat15 = FFFX
- Etat16 = FFFF

Parmi tous ces états possibles il y en a qui autorise le même traitement à effectuer sur le pixel courant. Ce sont les états 10 et 11 qui peuvent être assimilés au traitement de l'état 4.

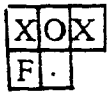
*) Nombre d'états où nco = 1

$$4 * 2^3 = 32$$

En fait, la manière dont on a défini l'algorithme, ne permet pas de fermer une chaîne en laissant une plus ancienne ouverte. Ce qui signifie qu'il y a un certain nombre

d'états impossibles.

Exemples de cas impossibles :



On recense 12 cas de ce genre.

On n'a que (32 - 12) = 20 états possibles où nco = 1. (voir tableau 1 pour la liste complète des états) Ces 20 états possibles présentent en fait 17 traitements différents. En effet, la mise en place de la filiation n'est pas faite systématiquement avec toutes les chaines du passé. C'est ainsi que le traitement de l'état 27 s'assimile à celui de l'état 26, de même pour l'état 35 avec l'état 32.

En détaillant un peu plus on se rend compte que seuls 7 états parmi eux nécessitent de tester le futur.

*) Nombre d'états où nco = 2.

En tenant compte de la remarque précédente et du fait que l'algorithme ne permet pas non plus d'avoir plus de deux chaînes ouvertes contigües on obtient 5 états possibles. Parmi ces 5 états, deux présentent des traitements identiques (l'état 39 et 40)

En résumé :

On recense finalement 41 états (16+20+5) possibles, mais qui correspondent à 36 traitements différents. Sept de ces états nécessitent de tester sur le futur.

3.2.2.2. Transitions de l'automate

Puisque nous voulons définir un automate nous devons définir les transitions entre états. Nous donnons dans le tableau 2 toutes les transitions de l'automate. Ces transitions sont très simples à définir comme indiqué ci-dessous :

- Exemple : de l'état 1 (XXXX) on peut aller sur les états (XX?O) où le "?" peut prendre toutes les valeurs possibles : O, X , F.
  Donc de l'état 1 on peut aller sur les états 17, 37, 22

Pour certains états, on est obligé de tester le futur.

- Exemple de l'état XXXO on peut aller sur :
  -> les états XX?O si le résultat du test est faux
  -> les états XX?F si le résultat du test est vrai

ETATS OU NCO = 0

Etat 1 = XXXX (Val_Et = 0)	;	Etat 2 = XXXF (Val_Et = 20)
Etat 3 = XXFX (Val_Et = 40)	;	Etat 4 = XFXX (Val_Et = 80)
Etat 5 = FXXX (Val_Et = 10)	;	Etat 6 = XXFF (Val_Et = 60)
Etat 7 = XFXF (Val_Et =100)	;	Etat 8 = FXXF (Val_Et = 30)
Etat 9 = FXFX (Val_Et = 50)	;	Etat 10 = FFXX (Val_Et = 90)
Etat 11 = XFFX (Val_Et =120)	;	Etat 12 = XFFF (Val_Et =140)
Etat 13 = FXFF (Val_Et = 70)	;	Etat 14 = FFXF (Val_Et =110)
Etat 15 = FFFX (Val_Et =130)	;	Etat 16 = FFFF (Val_Et =150)

Remarque : Etat 4 = 10 = 11

ETATS OU NCO = 1

Etat 17 = XXXO (Val_Et = 2)	;	Etat 18 = XXOX (Val_Et = 4)
Etat 19 = XOXX (Val_Et = 8)	;	Etat 20 = OXXX (Val_Et = 1)
Etat 21 = XXOF (Val_Et = 24)	;	Etat 22 = XXFO (Val_Et = 42)
Etat 23 = FXXO (Val_Et = 12)	;	Etat 24 = FOXX (Val_Et = 18)
Etat 25 = XFOX (Val_Et = 84)	;	Etat 26 = XFXO (Val_Et = 82)
Etat 27 = FFXO (Val_Et = 92)	;	Etat 28 = OXFX (Val_Et = 41)
Etat 29 = FXOX (Val_Et = 14)	;	Etat 30 = FXOF (Val_Et = 34)
Etat 31 = XFOF (Val_Et =104)	;	Etat 32 = XFFO (Val_Et =122)
Etat 33 = FXFO (Val_Et = 52)	;	Etat 34 = FFOX (Val_Et = 94)
Etat 35 = FFFO (Val_Et =132)	;	Etat 36 = FFOF (Val_Et =114)

Remarque : Etat 26 = 27 et 32 = 35
Les états 17, 18, 19, 20, 23, 24 et 25
nécessitent de tester le futur

ETATS OU NCO = 2

Etat 37 = XXOO (Val_Et = 6)	;	Etat 38 = FXOO (Val_Et = 16)
Etat 39 = XFOO (Val_Et = 86)	;	Etat 40 = FFOO (Val_Et = 96)
Etat 41 = OXOX (Val_Et = 5)		

Remarque : Etat 39 = 40

TABLEAU No 1 : Liste des états de l'automate

Les transitions possibles par état sont représentées par des triplets. Le nombre de triplet est soit de 1 ou de 2 sans test sur le futur, on parlera alors de transitions simples, soit de 2 ou de 4 avec test sur le futur,
on parlera alors de transitions composées (pour les états 17, 18, 19, 20, 23, 24 et 25).
En analysant un peu plus finement le tableau 2, nous pouvons remarquer que les triplets possibles sont en nombre restreint : 7 triplets possibles qui commencent respectivement par les états 2, 7, 8, 14, 17, 18, 23, 26. Leur fréquence respective est de 6, 12, 9, 8, 10, 1, 6, 6. Le triplet (17,37,22) représente donc la transition la plus fréquente.

La représentation du processus de création et de gestion des chaînes sous forme d'un automate à états finis avec des transitions sous forme de triplets doit faciliter la réalisation matérielle. Pour la réalisation logicielle d'un tel processus, nous proposons la mise en oeuvre décrite ci-dessous.

3.2.2.3. Détermination d'une valeur significative .

Il faut à partir des informations du passé, déterminer rapidement quel est le traitement à effectuer en fonction :

- le nombre de chaînes ouvertes (respectivement fermées) qui y conduisent
- la position des têtes ou queues de ces chaînes par rapport au pixel courant.

Le problème est qu'à chaque fois qu'on traite un pixel, il est nécessaire de connaître le type et la position de ses voisins, que ce soit pour inclure un point dans une chaîne ou bien pour mettre en place un lien de filiation.

Pour résoudre ce problème, l'idée est d'associer à chaque état de l'automate une valeur numérique non ambigue#. De cette manière, à chaque fois qu'on traite un pixel, on détermine la valeur significative de l'état, valeur qui permettra de se brancher directement à une procédure de traitement de cet état.

Pour réaliser cela simplement d'un point de vue logiciel, nous avons choisi de calculer un nombre dont le chiffre des unités codifie les chaînes ouvertes, les autres chiffres codifient les chaînes fermées. Ce nombre est de la forme 10*ncfv + ncov et se calcule de la façon suivante :

- On associe à chaque voisin du passé du pixel courant un poids .

1	8	2
4	.	

LISTE DES TRANSITIONS SIMPLES
de type 3 possibilités

De Etat 1	vers Etat	17, 37, 22	; De Etat 2	vers Etat	17, 37, 22
De Etat 3	vers Etat	26, 39, 32	; De Etat 4	vers Etat	23, 38, 33
De Etat 5	vers Etat	17, 37, 22	; De Etat 6	vers Etat	26, 39, 32
De Etat 7	vers Etat	8, 30, 13	; De Etat 8	vers Etat	17, 37, 22
De Etat 9	vers Etat	18, 39, 12	; De Etat 10	vers Etat	23, 38, 33
De Etat 11	vers Etat	23, 38, 33	; De Etat 12	vers Etat	14, 36, 16
De Etat 13	vers Etat	26, 39, 32	; De Etat 14	vers Etat	8, 30, 13
De Etat 15	vers Etat	26, 39, 32	; De Etat 16	vers Etat	14, 36, 16
De Etat 21	vers Etat	7, 31, 12	; De Etat 22	vers Etat	7, 31, 12
De Etat 26	vers Etat	8, 30, 13	; De Etat 27	vers Etat	8, 30, 13
De Etat 28	vers Etat	7, 31, 12	; De Etat 29	vers Etat	7, 31, 12
De Etat 30	vers Etat	7, 31, 12	; De Etat 31	vers Etat	14, 36, 16
De Etat 32	vers Etat	14, 36, 16	; De Etat 33	vers Etat	7, 31, 12
De Etat 35	vers Etat	14, 36, 16	; De Etat 37	vers Etat	7, 31, 12
De Etat 38	vers Etat	7, 31, 12	; De Etat 39	vers Etat	14, 36, 16
De Etat 40	vers Etat	14, 36, 16	; De Etat 41	vers Etat	7, 31, 12

LISTE DES TRANSITIONS SIMPLES
de type 6 possibilités

De Etat 34	vers Etat	8, 30, 13	ou Etat	14, 36, 16
De Etat 36	vers Etat	8, 30, 13	ou Etat	14, 36, 16

LISTE DES TRANSITIONS COMPOSEES
6 ou 12 possibilités

	Test futur faux		Test futur vrai
De Etat 17	vers Etat 17, 37, 22	--	vers Etat 2, 21, 6
De Etat 19	vers Etat 17, 37, 22	--	vers Etat 2, 21, 6
De Etat 20	vers Etat 17, 37, 22	--	vers Etat 2, 21, 6
De Etat 23	vers Etat 17, 37, 22	--	vers Etat 2, 21, 6
De Etat 24	vers Etat 17, 37, 22	--	vers Etat 2, 21, 6
De Etat 18	vers Etat 17, 37, 22	--	vers Etat 2, 21, 6
	ou Etat 26, 39, 32		ou Etat 7, 31, 12
De Etat 25	vers Etat 23, 38, 33	--	vers Etat 8, 30, 13
	ou Etat 26, 39, 32		ou Etat 14, 36, 16

TABLEAU No 2 : Liste des transitions de l'automate

- On associe la valeur 0 aux pixels qui sont du type 'X' et la valeur 1 aux autres types.
- Calcul de ncfv : on considère les pixels des types 'P' et 'O' comme étant du type 'X'. Alors ncfv est la somme des valeurs des pixels de la configuration pondérée par les poids des cases.
- Calcul de ncov : on fait de même avec les pixels de 'D' et 'F' comme étant de type 'X'.

Exemples :

F	O	X
X	.	

$10 * (1+0+0+0) + (0+8+0+0) = 18$

X	X	O
P	.	

$10 * (0+0+0+0) + (0+0+4+2) = 6$

L'ensemble des valeurs des états est indiqué dans le tableau 1 par le code Val_Et. Ce type de solution logicielle deviendrait très efficace en la modifiant légèrement pour une implémentation matérielle (exemple : par tabulation).

3.3. CONCLUSION SUR LE CHAINAGE

On vient de définir un algorithme de chaînage rapide qui permet de traiter tout type d'image présentant des contours d'épaisseur 1 et quelques cas particuliers de contours d'épaisseur 2, en un seul passage sur l'image.

L'algorithme est basé sur des concepts simples mais efficaces. Nous avons démontré que la combinatoire possible de tous les cas particuliers peut se ramener à la construction d'un automate à 41 états. Ces 41 états se décomposent en 36 traitements différents avec seulement 7 états où il est nécessaire de tenir compte du futur.

Cet algorithme est facile à mettre en oeuvre, le traitement cas par cas permettant d'obtenir un code efficace et modifiable sans trop de peine.

Toute l'information nécessaire aux traitements ultérieurs a été sauvergardée, en particulier tous les liens de filiation sont détectés et stockés.

En pratique, les résultats obtenus sont très satisfaisants. A titre d'exemple, le chaînage d'une image de 5000 pixels significatifs prend environ 6 secondes de temps CPU sur un Perkin Elmer 3250 (Unix système V) alors qu'il faut 3.8 secondes pour une image de zéros. Nous renvoyons le lecteur au chapitre VII, où nous présentons une implémentation parallèle possible de ce processus ainsi que des résultats sur différents jeux d'essais.

Bien sur cette étape n'est pas optimale au sens du nombre minimal de chaînes, et donc doit être suivi par une étape de fusion. Mais cette étape va travailler maintenant sur les chaînes et en particulier sur les en-tête de chaînes, où l'on a regroupé toute l'information nécessaire, et non plus sur l'image. C'est cette étape que nous allons maintenant développer.

4. FUSION DES CHAINES

L'exploration de l'image ligne à ligne à l'aide d'une fenêtre 3x3 en un seul passage sur l'image a pour conséquence de scinder certaines chaînes en plusieurs morceaux alors qu'aucune jonction de type "point triple" n'était présente. Un exemple caractéristique est donné sur la figure 5, entre les chaînes X et Y.

La fusion de chaînes est une opération qui consiste à former des chaînes de plus en plus longues sans porter préjudice aux jonctions, c'est à dire aux points triples ou quadruples, comme sur l'exemple avec les chaînes A, B et X. Elle permet la détection des chaînes qui se referment sur elles-mêmes, détectant ainsi un contour fermé.

Le principe général de l'algorithme de fusion peut s'exprimer ainsi :

Définition 8 :

- Soient deux chaînes X et Y, telle que
  X n'a qu'un fils Y et Y n'a pas d'autre aieul que X
- alors X et Y peuvent fusionner. La nouvelle chaîne est désignée par X.

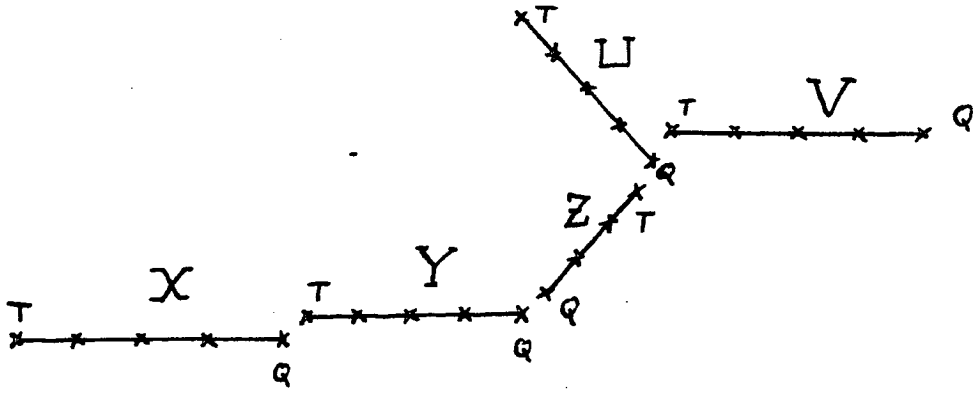
La fusion consiste alors à ajouter les maillons de Y aux maillons de X et à actualiser l'en-tête : gestion des liens de Y qui deviennent des liens de X, actualisation des informations générales.

Nous allons maintenant expliciter plus en détail l'algorithme.

4.1. CRITERE DE FUSION

Nous donnons ci-dessous un petit exemple. Après la création des chaînes on se trouve dans la situation suivante :

T = tête de la chaîne , Q = queue de la chaîne



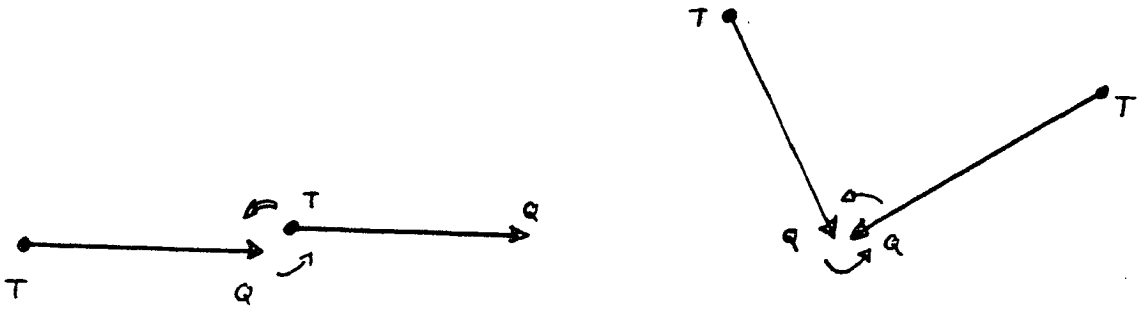
- figure 5 -

Principe de prolongement

Une chaîne X est prolongeable par la queue à l'aide d'une chaîne Y ssi :

(1) X a un seul fils Y

(2) X est le seul aieul de Y (lien Q-T) ou
X est le seul fils de Y (lien Q-Q)



aieul : ⇒ fils : →

Les mêmes critères sont utilisés pour la fusion sur tête.

Principe général de l'algorithme

On parcourt les chaînes par ordre de création. Soit X une chaîne. On boucle sur chacune des extrémités de X, d'abord la queue puis la tête. Pour l'extrémité A de X on essaye de fusionner la chaîne Y en lien avec X, les liens de Y deviennent les liens X et on continue jusqu'à ce que aucune fusion ne puisse être réalisée. Chaque fusion élimine de la liste des chaînes celle que l'on vient de fusionner.

Lorsque le traitement de X est terminé on recommence avec une autre chaîne de la liste restante et qui n'est pas encore prise en compte, jusqu'à épuisement de la liste.

Problème restant à traiter

Pour garder une connexité entre tous les maillons d'une chaîne finale, nous devons prêter attention à la manière d'ajouter les maillons de la chaîne Y dans la chaîne X. On fusionne toujours une paire d'extrémités de type { T , Q }. Or dans l'exemple précédent, nous avons une paire { Q, Q } pour la fusion de Y dans X. La solution consiste alors à effectuer un miroir de la chaîne Y avant de fusionner dans la chaîne X.

Il existera donc deux façons de procéder une façon directe et une façon miroir, en fonction des types de liens que présentent les chaînes. Ces types peuvent être :

- queue de X sur tête de Y - fusion directe.
- queue de X sur queue de Y - fusion avec miroir de Y
- tête de X sur queue de Y - fusion avec miroir de X
- tête de X sur tête de Y - fusion avec miroir de Y

4.2. MISE EN OEUVRE DE L'ALGORITHME DE FUSION

Nous présentons la mise en oeuvre de l'algorithme de fusion ainsi que toutes les fonctions nécessaires à son exécution (nous les développerons sous forme de pseudo-code lorsque nécessaire).

Si nous désignons par **kbuf** l'ensemble des chaînes à traiter, la fonction SUIVANT(kbuf) permet d'accéder à tous les élements de cet ensemble.

```
fonction FUSION (kbuf)
  -- Elle produit des effets de bord sur kbuf et ses éléments.

  début
      X ← SUIVANT(kbuf) ;
      tant que (X ≠ NIL) faire
            tant que (X est prolongeable en queue par la chaîne Y) faire
                  inclure Y dans Y ;
                  supprimer Y de kbuf ;
            fintq ;
            tant que (Y est prolongeable en tête par la chaîne Y) faire
                  inclure Y dans X ;
                  supprimer Y de kbuf ;
            fintq ;
            x ← SUIVANT(kbuf) ;
      fintq ;

fin FUSION ;
```

Comme on le voit l'algorithme de fusion des chaînes repose sur deux primitives principales :

** X est prolongeable en queue (respectivement tête) par Y
** inclure Y dans X.

4.2.1. Prolongement d'une chaîne.

Ce test est réalisé, en suivant la définition, par la combinaison de la fonction NBFILS (respectivement NBAIEUL) pour le prolongement en queue (respectivement tête) et la fonction UN_SEUL_LIEN.

Les fonctions NBFILS et NBAIEUL permettent de déterminer la chaîne Y et retournent si la chaîne X est un contour fermé. La fonction UN_SEUL_LIEN précise quelle est l'extrémité de Y qui est liaison avec X et retourne si à ce point la chaîne Y est en liaison seulement avec X.

```
fonction NBFILS( X )

  début
     si X∈ lfils( X) alors
              X est un contour ;
              return(NIL) ;
     finsi ;
     si (lfils(X) contient un seul élément Y) et (Y ≠ X) alors
              return (Y) ;
          sinon
              return (NIL) ;
     finsi ;

fin NBFILS ;
```

La fonction NBAIEUL(X) est similaire à NBFILS , pour les attributs de tête. Les fonctions lfils(X) et laieul(X) renvoient simplement la liste des numéros de chaînes qui sont connexes à X respectivement en queue ou en tête.

```
fonction UN_SEUL_LIEN (X , Y , extremite_Y) return boolean
  -- Cette fonction suppose que Y appartient à la famille de lien
     en tête ou en queue de X

  debut
     si (attribut de Y dans famille == 'T') alors
                extremite_Y ← 'T' ;
                si (X est le seul élément de laieul(Y)) alors
                           return(vrai) ;
                      sinon
                           return(faux) ;
                finsi ;
          sinon
                extremite_Y ∈ 'Q' ;
                si (X est le seul élément de lfils(Y)) alors
                           return(vrai) ;
                      sinon
                           return(faux) ;
                finsi ;
     finsi ;

fin UN_SEUL_LIEN ;
```

Donc X est prolongeable en queue par Y si l'on a :

(Y = NBFILS(X)) et (Y ≠ NIL) et (UN_SEUL_LIEN(X,Y,extremite_Y))

On obtient la prolongeabilité en tête de façon similaire.

4.2.2. Fonction d'inclusion

L'inclusion d'une chaîne dans une autre est une opération délicate parce qu'il faut régler les problèmes de cohérence, du parcours et des attributs de liens, de la chaîne résultante en fonction du type de chacune des extrémités de chaînes connexes, lorsque la fusion directe, mais surtout la fusion avec miroir a été effectuée.

L'algorithme d'inclusion nécessite les primitives suivantes :

- une primitive de changement de l'ordre des maillons dans une chaîne.
- une primitive qui résoud les problèmes de modifications de filiation aux extrémités qui changent de nom.
- une primitive qui permet de raccorder le maillon "queue" d'une chaîne au maillon "tête" d'une autre pour créer une chaîne plus longue.
- la gestion des modifications de filiation dues à une chaîne qui change de nom.

a) Primitive de changement de l'ordre des maillons d'une chaîne.

C'est la fonction MIROIR(X). C'est une fonction classique de changement d'ordre d'une liste chaînée qui effectue en plus la permutation de lfils(X) et de laieul(X).

b) Modifications dues au changement de nature d'extrémité d'une chaîne X.

La fonction INVERS_ORIENT réalise cette mutation pour toutes les chaînes Y qui sont liées a l'extrémité X. Désignons par famille l'ensemble des chaînes liées à Y par X (avec leur attribut).

```
fonction INVERS_ORIENT(famille,Y)

  début
    pour tout P ∈  famille faire
      si (attribut de P dans famille est 'D') alors
            chercher Y dans laieul(P) ;
            changer l'attribut de Y dans laieul(P) ;
        sinon
            chercher Y dans lfils(P) ;
            changer l'attribut de p dans lfils(P) ;
      finsi ;
    finpour ;

fin INVERS_ORIENT ;
```

c) Raccordement d'une chaîne Y en queue d'une chaîne X.

Il s'agit de concaténer le début de Y (maillon de tête) à la fin de X (maillon de queue). Ceci se réalise simplement en faisant pointer la queue de X sur la tête de Y. Après l'opération la chaîne X a pour fils ceux de Y avec les mêmes attributs. La primitive FUSCH(X,Y), qui assure cette fonction, retourne X.

d) Modifications de la filiation dues à une chaîne qui change de nom.

On remarque que la fonction FUSCH(X,Y) modifie l'environnement de la queue de Y. En effet, les chaînes qui étaient liées à Y par cette extrémité doivent maintenant être liées à X par un lien identique (voir Chap III, la double filiation). Il faut donc mettre à jour la filiation de chaînes constituant la filiation de Y devenue X. C'est le rôle de la fonction REMP_DANS_FAMILLE.

On désigne par **famille** l'ensemble des chaînes qui sont liées à l'extrémité considérée de Y. Soit Y_new = X.

```
fonction REMP_DANS_FAMILLE(famille,Y,Y_new)

  début
     pour tout Z ∈ famille faire
          si (attribut de Z dans famille est 'D') alors
                tenter de remplacer Y par Y_new dans laieul(Z) ;
                si on n'y arrive pas alors
                      laieuil(Y) ∈ laieul(Y) - {Y_new} ;
                finsi ;
           sinon
                tenter de remplacer Y par Y_new dans lfils(Z) ;
                 si on n'y arrive pas alors
                       lfils(Y) ∈ lfils(Y) - {Y_new} ;
                 finsi ;
          finsi ;
     finpour ;

fin REMP_DANS_FAMILLE ;
```

fonction d'inclusion de Y dans X

Ayant à notre disposition toutes ces primitives nous pouvons écrire la fonction INCLURE(X , extremite_X , Y , extremite_Y) qui retourne une chaîne. La chaîne X est prolongeable en "extremite_X" par Y qui présente à X son "extremite_Y".

```
fonction INCLURE(X , extremite_X , Y , extremite_Y) return chaîne

  début
     si (extremite_X = extremite_Y) alors
                        RETOURNER(Y) ; -- Combinaison de MIROIR et
                                       -- INVERS_ORIENT
     finsi;
     si (extremite_X = 'Q') alors
                        FUSCH(X,Y) ;
                        Installer la nouvelle valeur de X à la place
                              de l'ancienne dans kbuf ;
                        return(X) ;
          sinon
                        FUSCH(X,Y) ;
                        Installer Y à la place de X dans kbuf ;
                        return(Y) ;
     finsi;

fin INCLURE ;
```

Dans la fonction FUSION, "inclure Y dans X" est réalisé par

- X <- INCLURE(X,'Q',Y,extremite_Y) et
  REMP_DANS_FAMILLE(lfils(X),Y,X) ;
  ou
- X <- INCLURE(X,'T',Y,extremite_Y) et
  REMP_DANS_FAMILLE(laieul(X),Y,X) ;

suivant que X est prolongeable en queue ou en tête par Y. extremite_Y est donné par le test UN_SEUL_LIEN.

4.3. CONCLUSION

Nous venons de montrer, en détail, l'algorithme de fusion. Il permet de traiter tous les cas de fusions possibles avec une gestion efficace et rapide des problèmes de mise à jour des en-têtes et des maillons.

En pratique, les temps CPU obtenus pour la fusion sont tout à fait satisfaisant. Ils ne représentent qu'une petite partie du temps total pour le traitement complet (de l'ordre de 5 %). Nous renvoyons le lecteur au chapitre VII, pour un complément d'information sur les résultats.

L'étape de chaînage et l'étape de fusion représentent un algorithme complet de chaînage sur image de contour et nous aurions pu nous arrêter ici. Cependant dans la plupart des cas pratiques, les images de contour obtenues sont bruitées. Le bruit est soit du bruit isolé soit des barbules sur les contours. Comme nous désirons obtenir les chaînes les plus longues et les plus propres possibles, nous avons défini un algorithme d'élimination de chaînes que nous allons expliciter maintenant.

5. ELIMINATION DES CHAINES-BRUIT

Après la création et la fusion des chaînes, il nous reste le troisième type de traitement à définir : c'est l'élimination de chaînes qui sont considérées comme **parasites** (ou bruit).

On peut considérer comme bruits de chaînes, de petites chaînes isolées ou bien encore les barbules souvent rencontrées lors de la détection de contours. La difficulté est ici de ne pas confondre une chaîne qui peut représenter une information importante, et du bruit : par exemple une petite chaîne qui établit les liens entre deux groupes de chaînes ne doit pas être éliminée.

Le critère d'élimination que nous choisissons ici est **la longueur** de la chaîne. La longueur d'une chaîne est définie comme le nombre de maillons.

On aurait pu prendre un autre critère comme la moyenne des niveaux de gris, ou tout autre critère portant sur une information globale de la chaîne. Le principe de l'algorithme défini ci-dessous reste inchangé quelque soit le critère utilisé.

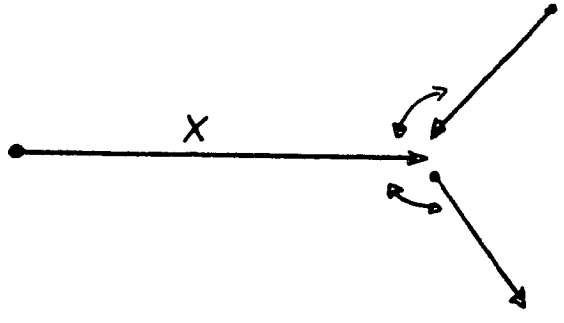
5.1. PRINCIPE

Définition 9 :

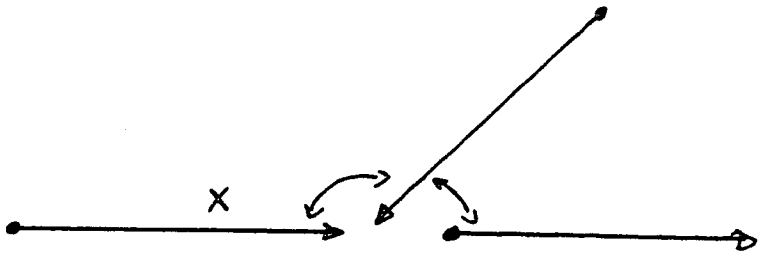
Nous dirons qu'une chaîne X est faiblement liée si :

(1) elle est isolée (ses deux extrêmités sont libres),

(2) ou bien l'une des extrêmités est libre et l'autre a au moins deux fils ou aieux.



(3) ou bien la chaîne X a une extrêmité libre et l'autre est liée à une seule chaîne Y qui elle-même, et sur la même extrémité, est liée à au moins une chaîne distincte de X.



Principe

On se donne un entier positif S que nous appelerons niveau de bruit. Toute chaîne X faiblement liée de longueur inférieure à S est appelée bruit. Nous verrons plus loin le cas spécial d'un contour fermé.

5.2. MISE EN OEUVRE DE L'ALGORITHME

L'algorithme de suppression d'un bruit X comporte quatre parties essentielles :

(1) Test pour déterminer si la chaîne X est faiblement liée.

(2) Suppression du lien des chaînes liées avec X.

(3) Mise à jour de la filiation des chaînes liées à X.

(4) Cas spécial de la chaîne X qui se referme sur elle-même.

5.2.1. TEST DE FAIBLE LIAISON

Soit X la chaîne à tester. On désigne par :

- **lfils(X)** l'ensemble des fils de la chaîne X. Chaque chaîne est munie de son attribut qui précise si elle est liée vers une tête 'T' ou queue 'Q'.
- **laieul(X)** est la fonction duale pour les aieux de X.
- **nbf(X)** = card(lfils(X))
- **nba(X)** = card(laieul(X)).

La chaîne X est isolée (donc faiblement liée) si (nbf(X) = nba(X) = 0).

Il reste alors deux situations à analyser :

** nbf(X) = 0 et nba(X) > 0
** nbf(X) > 0 et nba(X) = 0 (cas symétrique).

Nous traitons ici par exemple le premier cas. C'est le rôle de la fonction **TSUPTEST.**

```
fonction TSUPTEST(X) return boolean
   -- nbf(X) = 0 et nba(X) > 0

        début
          si (nba(X) > 1) alors
                    return (vrai) ;
          sinon
              -- nba(X) = 1
              {Y} <- laieul(X) ;
              si (attribut de Y dans laieul(X) est 'T') alors
                        si (nba(Y) = 1) alors
                              return (faux) ;
                          sinon
                              return(vrai) ;
                        finsi ;
                   sinon
                        si (nbf(Y) = 1) alors
                              return (faux) ;
                          sinon
                              return (vrai) ;
                        finsi ;
              finsi ;
          finsi ;

fin TSUPTEST ;
```

5.2.2. SUPPRESSION DU LIEN DANS LES CHAINES CONNEXES AVEC X.

On garde les mêmes notations que ci-dessus. Il s'agit de parcourir la famille correspondant à l'extremité de X considéré. Pour chaque chaîne Y de cette famille on supprime son lien avec X. C'est le rôle de la fonction **SUP_DANS_FAMILLE.**

```
fonction SUP_DANS_FAMILLE(X,famille)

  début
     pour tout Y ∈ famille faire
           si (attribut de Y dans famille est 'T') alors
                    laieul(Y) ← laieul(Y) - {X} ;
              sinon
                    -- X ∈ lfils(Y)
                    lfils(Y) ← lfils(Y) - {X} ;
           finsi
     fin pour ;

  end SUP_DANS_FAMILLE
```

5.2.3. MISE A JOUR DES FILIATIONS DES CHAINES CONNEXES A X .

C'est une partie délicate de l'algorithme de suppression de bruit. L'idée de l'algorithme est de choisir une chaîne privilégiée de la liste **famille** et de la lier à toutes celles qui restent en respectant les règles de filiation.

Nous construisons alors la fonction **REFILIAT** qui suit :

```
fonction REFILIAT (famille)

        début
            P ← PREMIER(famille) ;
            -- produit le premier élément de "famille"
            si (attribut de P dans famille est 'T') alors
                pour tout Y ∈ (famille - {P}) faire
                        laieul(P) ← laieul(P) + {Y} ;
                        si (attribut de Y dans famille est 'T')
                                laieul(Y) ← laieul(Y) + {P} ;
                        sinon
                                lfils(Y) ← lfils(Y) + {P} ;
                        finsi ;

                finpour ;
            sinon
                pour tout Y ∈ (famille - {P}) faire
                        lfils(P) ← lfils(P) + {Y} ;
                        si (attribut de Y dans famille est 'T')
                                laieul(Y) ← laieul(Y) + {P} ;
                        sinon
                                lfils(Y) ← lfils(Y) + {P} ;
                        finsi ;
                finpour ;
            finsi ;

    fin REFILIAT ;
```

Remarques sur l'établissement des filiations .

1/ De façon générale nous avons pris soin de ne pas lier plus d'une fois l'extrêmité d'une chaîne à une autre même chaîne.
2/ A la génération d'une chaîne contenant un seul pixel, toutes les liaisons ont été faites en tête. Ce n'est donc pas un cas particulier.

5.2.4. CAS PARTICULIER DES CHAINES QUI SE REFERMENT SUR ELLES-MEMES.

Ce cas est facile à traiter. Si X est une chaîne qui se referme sur elle-même il suffit de vérifier que :

$X \in lfils(X)$ et $X \in laieul(X)$ et longueur(X) < S

5.2.5. ORGANISATION GENERALE DE L'ALGORITHME

L'algorithme de suppression de bruit se présente donc comme suit:

```
fonction BRUITAGE( kbuf , S)
        - kbuf désigne l'ensemble des chaines à traiter.
        - S niveau de bruit

  début

     pour tout X ∈ kbuf faire

            si (X est isolée) et (LONGUEUR( X) < S) alors
                 supprimer X de kbuf ;
            finsi ;

            si (nba(X) = 0) et (nbf(X) > 0) et TSUPTEST(X) et
                          (LONGUEUR(X) < S) alors
                 - la tête de X est libre
                 SUP_DANS_FAMILLE(lfils(X),X) ;
                 REFILAT(lfils(X)) ;
                 supprimer X de kbuf ;
            finsi ;
            si (nba(x) > 0) et (nbf(X) = 0) et TSUPTEST(X) et
                          (LONGUEUR(X) < S) alors
                 -- la queue de x est libre
                 SUP_DANS_FAMILLE(laieul(X),X) ;
                 REFILAT(laieul(X)) ;
                    supprimer X de kbuf ;
            finsi ;

            si ( X ∈ ((lfis(x) et laieul(x)) et (LONGUEUR(X) < S)
                 alors
                 - X est une boucle
                 SUP_DANS_FAMILLE(laieul(X),X) ;
                 SUP_DANS_FAMILLE(lfils(X),X) ;
                 REFILAT(laieul(X)) ;
                REFILAT(lfils(X)) ;
                 supprimer X de kbuf ;
            finsi ;

     finpour ;

fin BRUITAGE ;
```

6. PROPOSITION D'UN TRAITEMENT GLOBAL OPTIMISE

Notre but final est de représenter l'image avec un nombre minimum de chaînes sans perte d'information. Pour cela, nous avons décomposé le problème en trois fonctions plus simples. Il nous reste à voir maintenant comment utiliser de manière opérationnelle ces fonctions.

La remarque suivante permet de définir un algorithme optimal :

> Chaque élimination de bruit accroît la possibilité de fusion de chaînes. Deux chaînes qui pouvaient être considérées comme du bruit peuvent, après fusion, former une chaîne qui dépasse le seuil de bruit.

La méthode **optimale** suivant le critère de la longueur, consiste alors à faire :

> si nous voulons éliminer les bruits de niveau S, alors effectuer l'élimination progressive en itérant avec un seuil K incrémental, jusqu'à ce que K = S

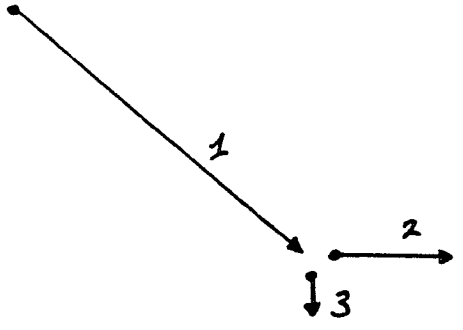
Cela nous permet de définir l'algorithme optimal permettant d'effectuer le chaînage de contour.

```
CREATION DES CHAINES

pour i jusqu'à S par pas de 1  faire
    FUSION(kbuf) ;
    BRUITAGE(kbuf,i) ;
finpour;

FUSION(kbuf) ;

MISE A JOUR DE LA NUMEROTATION DES CHAINES

ECRITURE SUR DISQUE
```

Nous terminons par FUSION afin d'exploiter les possibilités laissées par le dernier BRUITAGE de la boucle.

Le programme de mise à jour de la numérotation permet simplement d'avoir une liste compacte de numéros. Il s'agit essentiellement d'une mise à jour dans les liens de filiation.

Nous allons illustrer l'optimalité de cette méthode par un exemple. Considérons le schéma suivant :



LONGUEUR(chaîne 1) = 10
LONGUEUR(chaîne 2) = 9
LONGUEUR(chaîne 3) = 1

On choisit comme niveau de bruit S = 10 .

Si on commence brutalement par le niveau de bruit 10, la chaîne 1 est éliminée puis les chaînes 2 et 3 fusionnent en une nouvelle chaîne de longueur
9 + 1 = 10 qui peut être supprimée.

Ainsi l'ensemble est considéré comme du bruit alors qu'avec la méthode proposée, seule la chaîne 3 est éliminée et les chaînes 1 et 2 fusionnent pour une chaîne de longueur 19.

7. MISE EN OEUVRE PRATIQUE

7.1. RESULTATS

Nous avons regroupé en annexe les différents résultats obtenus sur divers types d'images. Les images ont toutes la même dimension initiale 256x256. Les temps CPU de l'algorithme de chaînage dépendent surtout du nombre de pixels significatifs.

C'est pourquoi, pour chacune des images on présente :

- L'image initiale.
- Une image qui montre le résultat du chainage et le nombre de points traités.
- Une image qui montre le résultat obtenu quand on effectue en plus une étape pour éliminer le bruit dont le seuil est fixé par l'utilisateur.

Les temps d'exécution indiqués correspondent aux résultats obtenus par un programme implémenté en C et qui s'exécute sur un Perkin-Elmer 3254 sous Unix système V. Ces temps ne tiennent pas compte des entrées sorties sur disques. La structure de données utilisée fait l'objet d'un rapport technique Inria [Gar 86].

7.2. COMPLEXITE ALGORITHMIQUE.

Nous allons détailler ci-dessous la complexité de l'algorithme en terme de place mémoire requise et d'instructions par pixel.

7.2.1. Place mémoire

On considère une image de taille NxN codée sur 1 octet.
Il faut deux lignes images pour effectuer le traitement, la ligne courante et la ligne future. On peut remarquer que pour éviter de trop fréquents appels disque on peut stocker des blocs de lignes. Cela dépendra de la place mémoire disponible par l'utilisateur. On regardera ici la place minimale nécessaire.
Il faut stocker le tableau ITAB qui est un tableau équivalent en place mémoire à quatre lignes images où les informations nécessaires doivent être codées en entier (stockage de pointeur).
Il faut un vecteur d'entier (kbuf) pour stocker la correspondance numéro de chaine - pointeur de chaine.
La structure de stockage des chaînes en mémoire centrale est la suivante :

une chaîne est une union de structures au sens C, composée d'un en-tête et de chaînons. Ces données sont réparties en mémoire de manière aléatoire selon le principe suivant :

en-tête	maillon
numero de chaine	coordonnée x
adresse du dernier maillon	coordonnée y
famille en tete	niveau de gris
famille en queue	adresse du suivant
adresse du premier maillon	

La place mémoire nécessaire est, bien sûr, fonction du nombre de pixels de contour de l'image (pixels significatifs) et du nombre de chaînes créées. D'un point de vue statistique sur les images que nous avons traitées, on peut dire que le nombre de pixels significatifs est de l'ordre de 10% des pixels de l'image brute, et qu'il faut compter, avant l'étape fusion-élimination, une chaîne pour sept pixels. Même si à la fin du traitement la place mémoire requise en beaucoup plus faible, il faut compter à un instant donné de l'algorithme disposer de :

(X * 4 + X/7 * 11) mots entiers,
où X = (N * N / 10) pixels

En résumé pour effectuer le chaînage, il faut :

(6 * N) + (X * 4) + (12 * X/7) mots entiers

Pour prendre un exemple si N = 256, alors il faut environ : **150 koctets** pour faire tourner l'algorithme.

7.2.2. Nombre d'instructions par pixel

La plus grande partie du temps CPU est consommé par le traitement de création des chaînes et lecture de l'image (plus de 90% du temps). C'est donc cette partie que nous allons expliciter, et plus précisément le nombre d'instruction pour un pixel une fois que la fenêtre 3x3 d'exploration est centrée sur lui.

Pour tous les pixels de l'image (significatifs et non significatifs), il faut effectuer un test pour déterminer sa classe.
Pour tous les pixels significatifs et la plupart du temps, d'un point de vue statistique, il faut tester le passé, inclure le pixel dans une chaîne et moins souvent tester le futur du pixel. Au chapitre III, un automate à états finis est proposé comme solution. Si cette solution est optimale vis à vis d'une implémentation matérielle, nous estimons cependant ici le coût d'une implémentation logicielle.
La détermination du traitement à faire (i.e. la détermination de l'état) coûte 7 additions

et une multiplication par pixel significatif. L'adjonction d'un pixel dans la chaîne se fait de manière immédiate par récupération de l'adresse mémoire de la chaîne dans le tableau ITAB (cf chapitre III), puis récupération de l'adresse mémoire du dernier maillon inclus précédemment, chargement de la nouvelle adresse du dernier maillon dans l'en-tête et dans ce que l'on considère maintenant comme l'avant dernier maillon.

En résumé, si nous nous plaçons d'un point de vue logiciel, et en ne tenant compte que des instructions fondamentales de l'algorithme, nous avons :

- une instruction par pixel image (test de validité sur vecteur indexé)
- environ 15 instructions par pixel significatif qui n'est pas une jonction potentielle (test et manipulation de données mémoire sur tableau indexé).
- environ 35 instructions par pixel représentant une jonction potentielle (test et manipulation de données mémoire sur tableau indexé).

Ce qui donne pour une image totale et en reprenant les mêmes chiffres que sur l'exemple précédent :

(N*N) + (15 * N*N/10) + (35 * N*N/70) instructions programmes
=> environ 3*N*N instructions programmes de type C

Pour une image 256x256 et en prenant pour base 10 microsecondes par instruction cela nous donne :

environ 2 secondes par image pour la partie chainage initial.

Ce temps ne tient pas compte du temps passé dans les appels de fonctions, dans la gestion des boucles et dans les appels images.

7.3. FAISABILITE MATERIELLE

Nous avons vu que l'algorithme se décompose en trois taches simples et distinctes que l'on rappelle ci-dessous :

1- Création et gestion des chaînes
2- Fusion des chaînes
3- Elimination des chaînes.

La nature des actions et des données à traiter est fondamentalement différente entre les taches 2 et 3 et la tache 1. Cette dernière est effectuée en une passe sur l'image. Ces données d'entrée sont donc les pixels qui arrivent au rythme vidéo. Cette tache est donc liée à une cadence précise de traitement. Au contraire, les taches 2 et 3 travaillent sur des listes, la seule contrainte qui leur est imposée est un délai de traitement. Typiquement, il faut qu'elles aient fini leur traitement sur l'image i lorsque la

création des chaînes sur l'image i+1 est disponible. La différence peut paraitre subtile, elle est néanmoins fondamentale :

- La tache 1 travaille à la cadence pixel : aucune attente n'est possible, le traitement du pixel k doit impérativement être fini à l'arrivée du pixel k+1. Or nous avons mesuré lors des simulations que justement c'est cette tache qui est la plus consommatrice de CPU.
- Les taches 2 et 3 travaillent à la cadence image, il y a désynchronisation entre l'arrivée des pixels et ces traitements : il faut que globalement les traitements soient finis à l'arrivée de la prochaine image. Ces taches, même si elles sont complexes, ne doivent être effectuées que sur un nombre de données limitées.

Analysons maintenant plus en détail la tache numéro 1. Nous pouvons, après une rapide analyse, décomposer celle-ci en trois parties :

a) Détection de l'état de l'automate à l'instant k.

b) Exécution des ordres de traitements liés à cet état.

c) Mise à jour des données du tableau ITAB.

Etudions maintenant :

de quels résultats du traitement du pixel à l'instant k a-t-on réellement besoin pour connaitre les ordres à effectuer pour le pixel de l'instant k+1 ?

La réponse est simple : de l'état de l'automate à l'instant k+1. Or celui-ci dépend de la mise à jour du tableau ITAB. Celle-ci se résume à attribuer au pixel courant à l'instant k la valeur "D","P","O" "F", à lui raccrocher un numéro de chaîne, et à changer éventuellement un type sur les pixels passés (de "O" en "F", de "D" en "P", etc...). Ceci doit donc être exécuté à la cadence pixel.

Cela signifie que nous faisons une grande distinction entre connaître les ordres à effectuer ("ajouter le maillon dans la chaîne X) et les exécuter. Regardons maintenant le type des ordres à exécuter lorsqu'un état de l'automate est connu. Nous décomposons ces ordres en quatre classes :

- Test du Futur : existence de un ou plusieurs pixels dans le futur.
- Ouverture d'une chaîne : allocation d'un numéro et d'un pointeur mémoire.
- adjonction d'un maillon dans une chaîne.
- Etablissement des liens de filiation.

Ici aussi il n'y a que les deux premières classes (les plus simples d'un point de vue matériel) qui doivent être réalisé à la cadence pixel. Les deux autres (les plus complexes) qui de plus sont indépendantes peuvent être mise dans deux piles (on parlera de pîle d'ordre) et donc être désynchronisées de la cadence pixel. L'idée est d'utiliser le temps laissé par le non-traitement des pixels non significatifs pour vider les pîles et donc réguler ainsi le débit. Exprimé de cette manière, le parallélisme intrinsèque de

l'algorithme devient une évidence. Nous donnons à titre d'exemple dans le tableau 3, les fréquences de chacun des états de l'automate pour la première image donnée en annexe

En résumé :

Au débit pixel : solution cablée

- l'automate à états finis
- mise à jour de ITAB
- obtention d'une numéro de chaîne (par incrément)
- test sur le futur

Au débit image : solution microprogrammée utilisant des mémoires tampons

- incorporation de maillons dans les chaînes
- mise en place des liens de filiation.
- fusion des chaînes
- élimination des chaînes

Une conception matérielle tenant le débit vidéo semble donc tout à fait réaliste par un système multiprocesseurs. On peut de plus remarquer la possibilité d'un fort parallélisme par découpage en bandes horizontales (exemple 16 bandes avec recouvrement d'une ligne, pour une image de 512 lignes), chaque bande étant traitée au niveau de la création des chaînes par des unités de traitement indépendantes. Une solution de type SPMD ('Single Program-Multiple Data') avec 16 processeurs (PEs) convient tout à fait pour la réalisation d'un tel processus. Les échanges se limitent à l'envoi par chaque PE de la première ligne une fois traitée au PE de la bande précédente. Un traitement spécial, à la fin de chaque bande permet la mise en place des liens de filiation avec les chaînes de la bande inférieure. Les processus de fusion et d'élimination s'effectuent ensuite sur l'ensemble des chaînes obtenues.

nombre de chaines est : 971
nombre de pixels traites : 4869

Frequence de l'etat 1	: 0.155268	<-
Frequence de l'etat 2	: 0.006572	
Frequence de l'etat 3	: 0.014787	
Frequence de l'etat 4	: 0.006572	
Frequence de l'etat 5	: 0.004929	
Frequence de l'etat 6	: 0.000822	
Frequence de l'etat 7	: 0.009858	
Frequence de l'etat 8	: 0.008421	
Frequence de l'etat 9	: 0.001027	
Frequence de l'etat 12	: 0.000411	
Frequence de l'etat 13	: 0.000000	
Frequence de l'etat 14	: 0.000000	
Frequence de l'etat 15	: 0.001027	
Frequence de l'etat 16	: 0.000000	
Frequence de l'etat 17	: 0.281577	<-
Frequence de l'etat 18	: 0.036763	
Frequence de l'etat 19	: 0.360649	<-
Frequence de l'etat 20	: 0.060998	
Frequence de l'etat 21	: 0.000000	
Frequence de l'etat 22	: 0.021565	
Frequence de l'etat 23	: 0.002670	
Frequence de l'etat 25	: 0.001438	
Frequence de l'etat 24	: 0.002054	
Frequence de l'etat 26	: 0.004313	
Frequence de l'etat 28	: 0.001848	
Frequence de l'etat 29	: 0.001232	
Frequence de l'etat 30	: 0.000205	
Frequence de l'etat 31	: 0.000205	
Frequence de l'etat 32	: 0.000205	
Frequence de l'etat 33	: 0.000000	
Frequence de l'etat 34	: 0.000822	
Frequence de l'etat 36	: 0.000000	
Frequence de l'etat 37	: 0.011707	
Frequence de l'etat 38	: 0.000000	
Frequence de l'etat 39	: 0.000411	
Frequence de l'etat 41	: 0.001643	

- Tableau 3 -

Remarque : Automate à 36 états .
Etat 4 = 10 = 11 , 26 = 27 , 32 =35 , 39 = 40.
Les états dont la fréquence dépassent 10% sont marqués par "<-"

8. CONCLUSION

Nous venons d'exposer un outil général pour toute mise en oeuvre d'un processus de Vision par Ordinateur, permettant le passage de la représentation matricielle à la représentation de type liste sans perte d'information. La représentation d'une image de contours sous forme de chaînes n'est pas un problème aussi simple qu'il pouvait le paraître. Notre décomposition du problème en trois parties distinctes (création des chaînes, fusion et élimination) nous a permis d'arriver à une solution optimale respectant l'ensemble des objectifs fixés :

- chaînage rapide en un seul balayage de l'image par une fenêtre glissante 3x3.
- gestion efficace des liens de parenté, par détection et stockage de toutes les jonctions de chaînes.
- élimination des chaînes suivant un critère de longueur ou tout autre critère portant sur une information globale de la chaîne.
- minimisation du nombre de chaînes, suivant ce critère, par une combinaison itérative de fusion-élimination.
- possibilité de stockage étendu pour divers types d'information (niveau de gris pixel, amplitude gradient , etc..)
- stockage image réduit puisque l'agorithme nécessite la présence de deux lignes image seulement.
- faisabilité temps réel démontrée et envisageable pour un coût réduit.

Notre méthode, très générale, réalise l'ensemble de ces objectifs pour n'importe quelle image de type contour sous contrainte d'avoir des contours d'épaisseur 1 presque partout.

Nous avons proposé pour la phase la plus délicate de la méthode (la création des chaînes) une solution mettant en oeuvre un automate à 41 états. Une solution matérielle réaliste a été proposé et est en cours de développement, utilisant le parallélisme asynchrone naturel du processus. En effet, le processus se décompose en deux parties comme indiqué ci-dessous :

- Choix du traitement à effectuer par pixel. Ce choix doit être effectué pour chaque pixel de l'image, donc lié au rythme de la cadence d'acquisition pixel. Nous résolvons ce problème par l'utilisation de l'automate à états finis
- Exécution du traitement. Seul un petit nombre de pixels de l'image nécessite l'exécution d'un traitement, ce sont justement les pixels de contour. Cette exécution n'est donc pas liée au rythme de l'acquisition pixel mais à celui beaucoup plus lent de l'obtention d'une trame vidéo. Les traitements à effectuer qui sont de deux types, peuvent être exécutés eux aussi dans un parallélisme asynchrone. Nous proposons une solution microprogrammée en utilisant deux piles :

* une pile concernant la gestion de la chaîne : ajoût d'un élément, fermeture etc...
* une pile concernant la gestion des filiations : mettre un lien de tête de la chaîne X sur la chaîne Y, etc...

La seule amélioration qu'il serait intéressant d'effectuer est l'adjonction d'autres critères d'élimination en particulier liés à la moyenne des niveaux de gris de la chaîne. Exemple d'utilisation :

soit une image pseudo binaire où les pixels significatifs sont les maxima locaux valués d'une image de gradient. Un critère d'élimination sur la moyenne des amplitudes de chaque chaîne permet d'obtenir un seuillage plus performant que le simple seuillage sur chaque pixel.

Remerciements : Je tiens particulièrement à remercier ici Patrick Cipière, ingénieur système du projet qui a codé un certain nombre de fonctions de base pour la réalisation du programme général, ainsi que Philippe Garnesson et Aimé Medenouvo qui ont participé, au cours d'un stage de DESS [Gar 86], à l'élaboration de l'algorithme final en particulier par la création d'un programme générant automatiquement le code de l'automate à états finis.

REFERENCES

[Cha 81]

Indranil CHAKRAVARTY : 'A single-Pass, Chain Generating Algorithm for Region Boundaries', in Computer Graphics and Image Processing, Vol 15, 1981, pp 182-193.

[Fre 74]

H. FREEMAN : 'Computer Processing of Line Drawing images', in Computing Surveys, Vol 6, 1978, pp 57-97.

[Gar 86]

P. GARNESSON et A. MEDENOUVO : 'Un Algorithme Rapide de Chainage ', Rapport de projet du DESS de l'I.S.I., Mai 1986.

[Kul 74]

A. K. KULKANI : 'Sequential Shape Feature Extraction From Line-Drawings', IEEE Computer Society Conference, PRIP, Chicago, Illinois, May 1978, pp 230-237.

[Min 69]

M. MINSKY and S. PAPERT : 'Perceptrons, An Introduction to Computatinal Geometry', MIT Press, Cambridge, Mass 1969.

[Pav 78]

T. PAVLIDIS : 'A Minimum Storage Boundary Tracing Algorithm and its Application to Automatic Inspection', in IEEE Trans Syst Man Cybern, SMC-8, 1978, pp 66-69

[Ros 70]

A. ROSENFELD : 'Connectivity in Digital Pictures', in Journal of ACM, Vol 17, January 1970, pp 146-160.

[Ros 78]

A. ROSENFELD : 'Algorithms for Image/Vector Conversion', in Computer Graphics Vol 12, August 1978, pp 135-139.

[Sel 72]

S. SELKOW : 'One-Pass Complexity of Digital Picture Propertries', In Journal of ACM, Vol 19, April 1972, pp 283-295.

[Sob 78]

I. SOBEL : 'Neighborhood Coding of Binary Images for Fast Contour Following and General Array Binary Processing', in Computer Graphics and Image Processing, Vol 8, 1978, pp 127-135.

ANNEXE

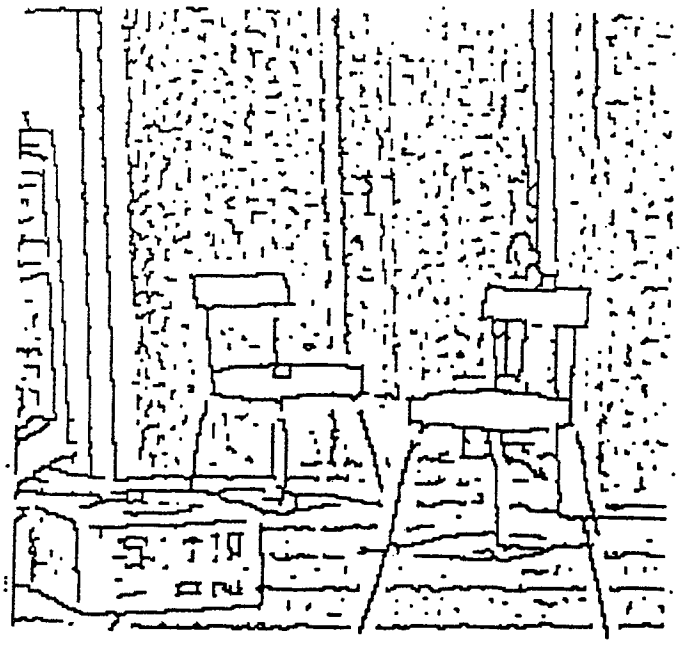
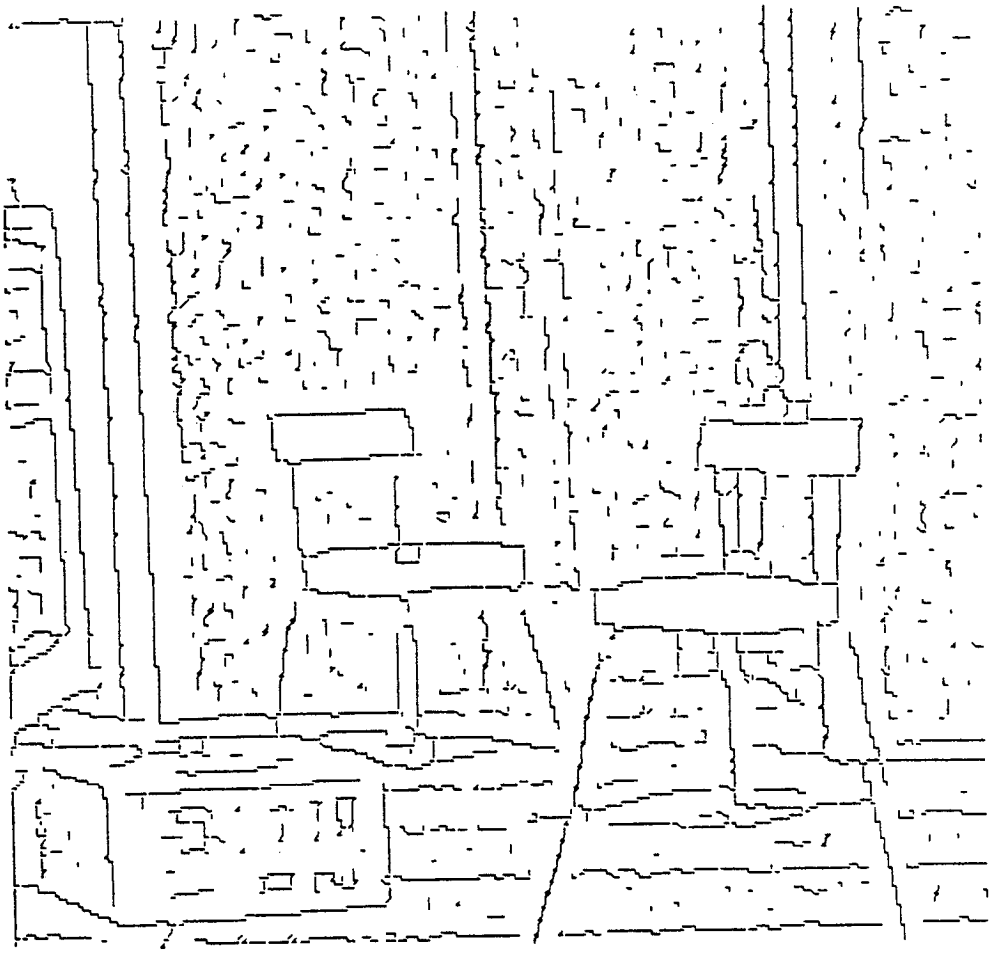


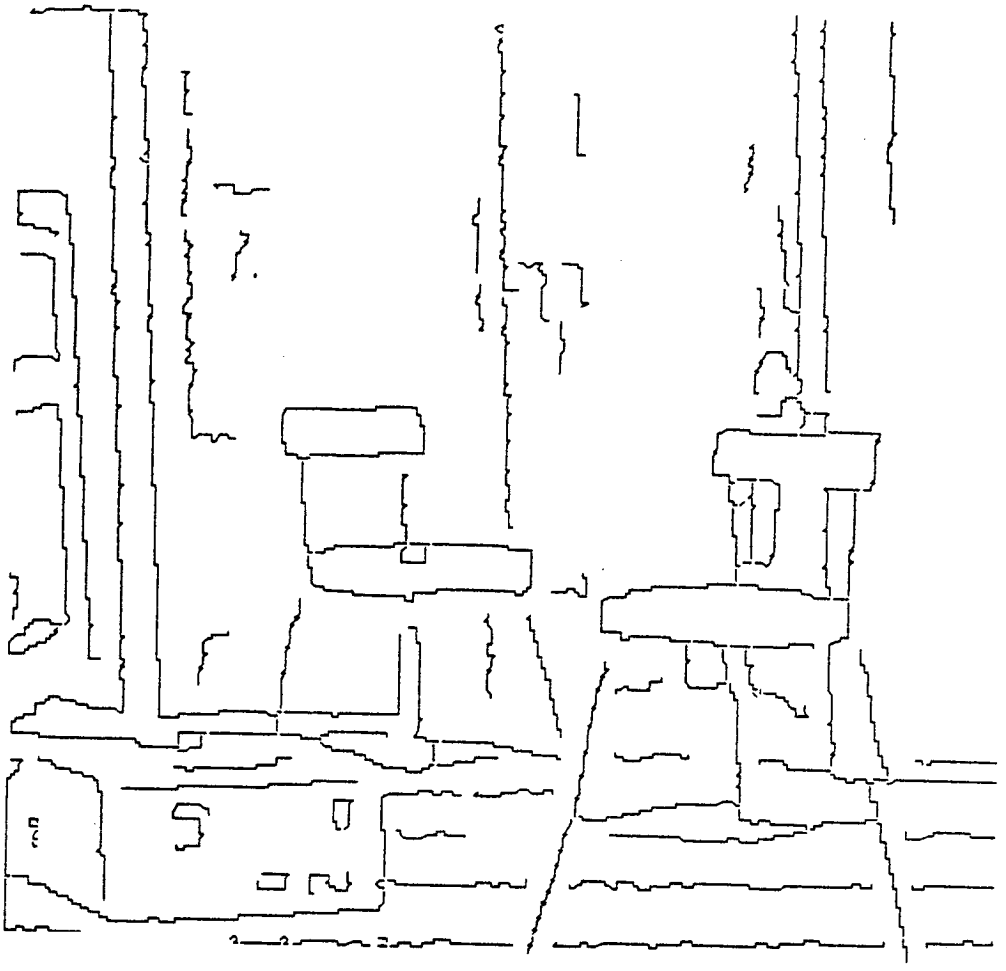
Image Brute : 256x256



Traitement : chainage seul

Nombre de pixels traités : 7487
Nombre de chaines créées : 1475

Temps d'exécution 6.46 secondes CPU



Traitement complet

Seuil du Bruit = 15

Nombre de chaines fusionnées = 480
Nombre de chaines éliminées = 855
Nombre de chaines restantes = 140

Temps d'exécution 9.6 secondes CPU

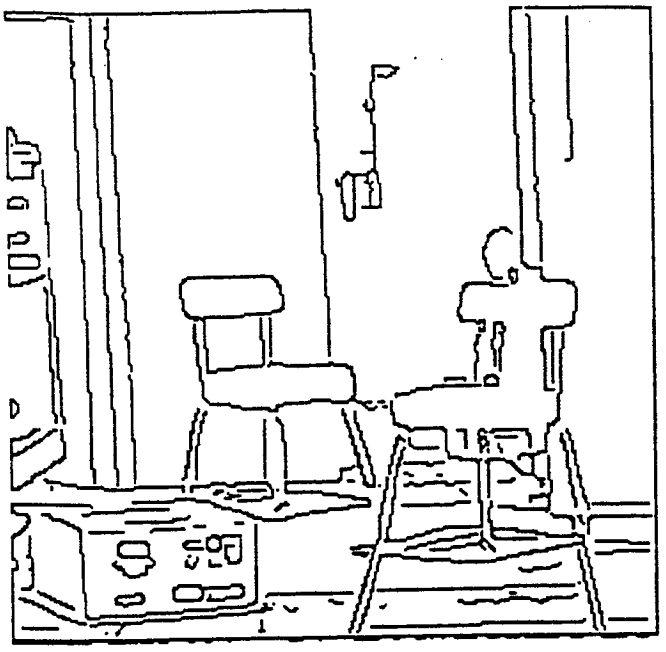
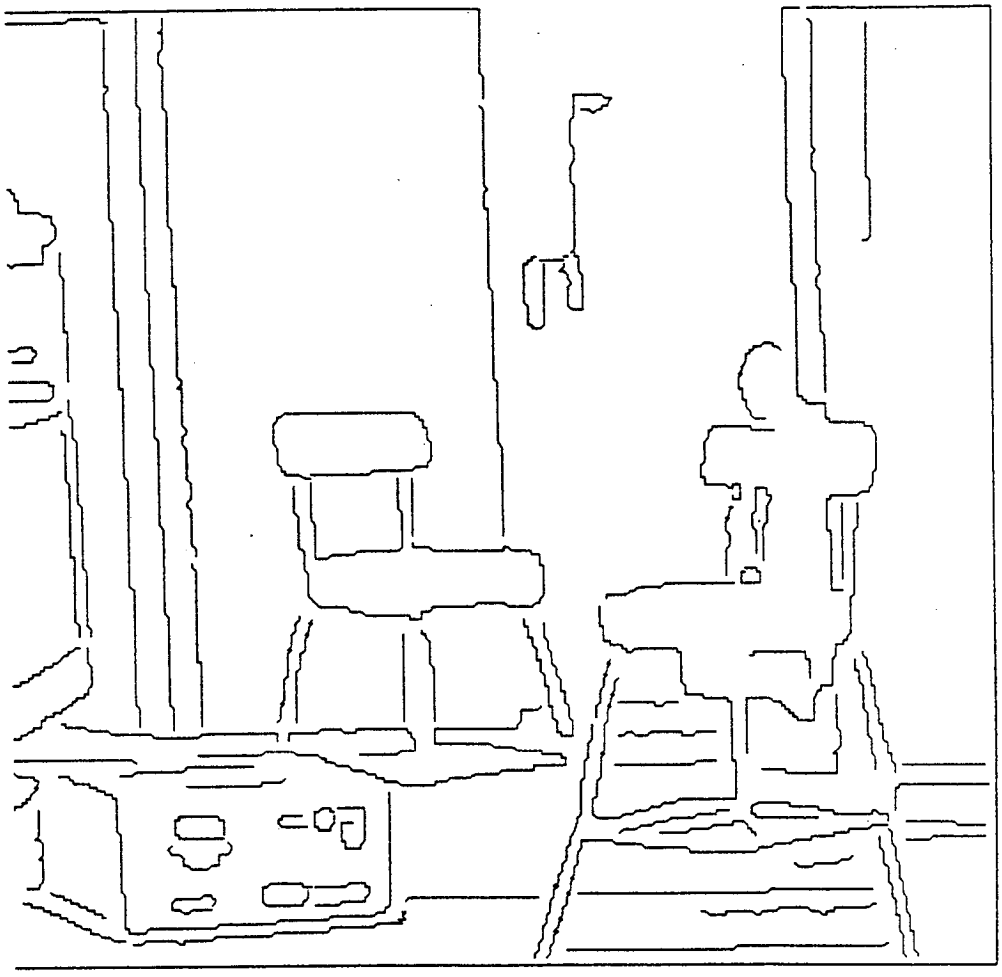


Image Brute : 256x256

Traitement : chainage seul

Nombre de pixels traités : 5599
Nombre de chaines créées : 495

Temps d'exécution 6.2 secondes CPU



Traitement complet

Seuil du Bruit = 15

Nombre de chaines fusionnées = 301
Nombre de chaines éliminées = 107
Nombre de chaines restantes = 87

Temps d'exécution 8.50 secondes CPU

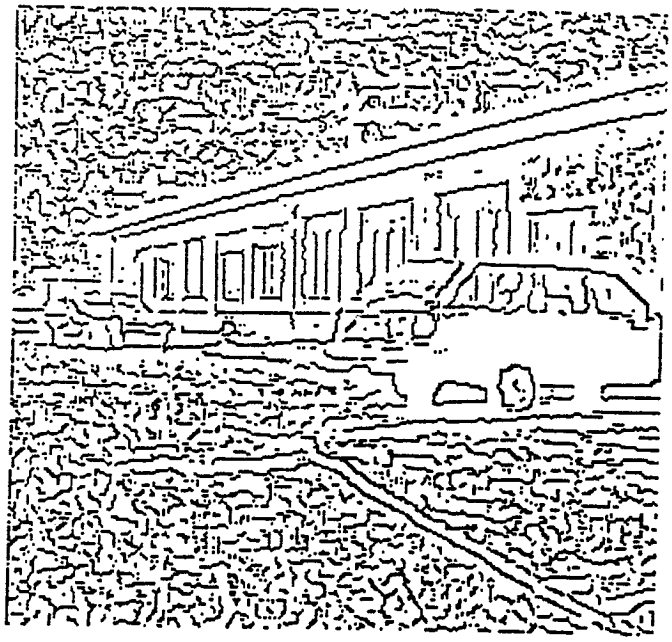
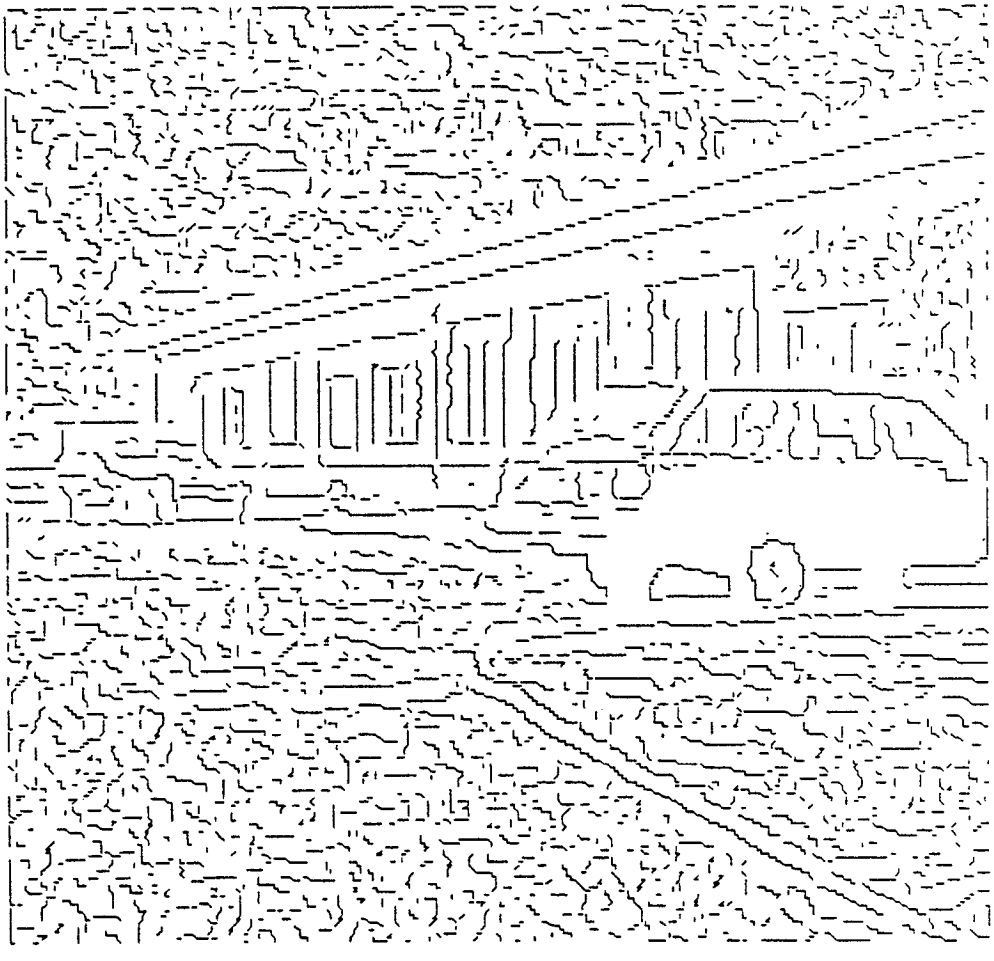


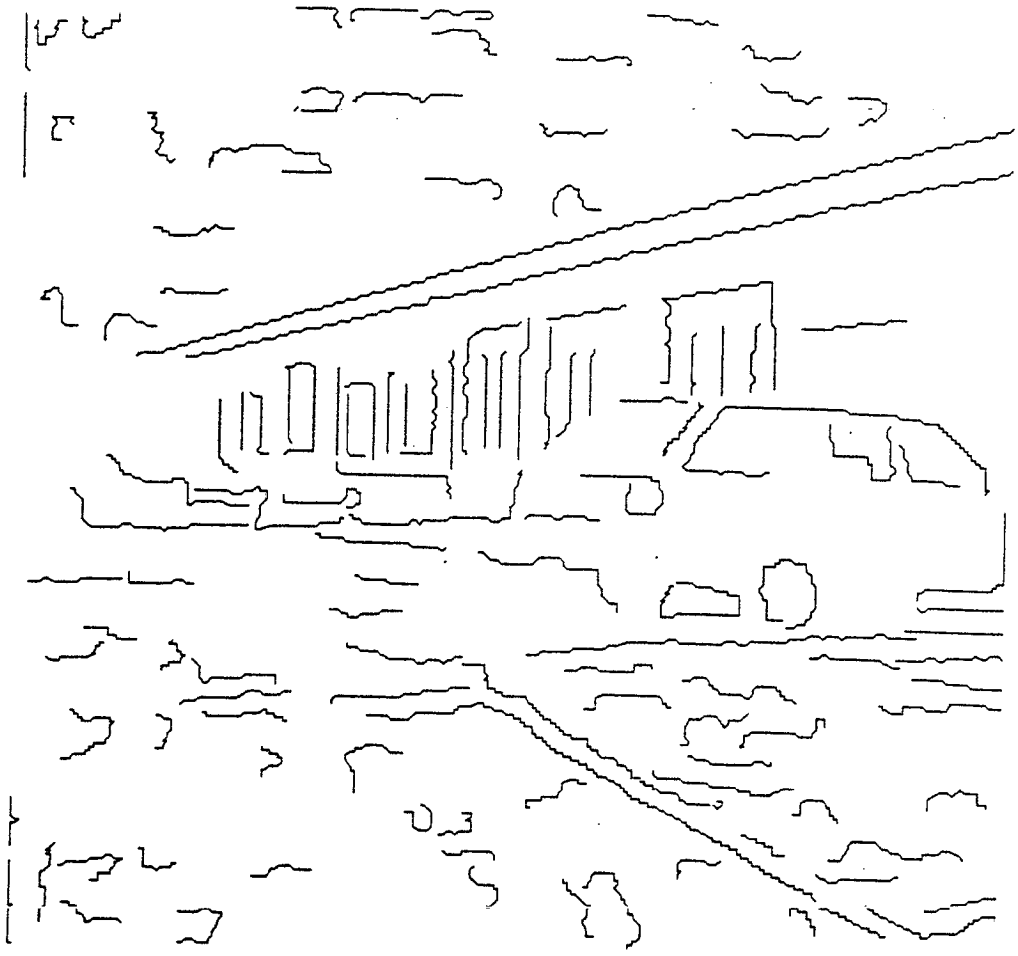
Image Brute 256x256



Traitement chainage seul

Nombre de pixels traités : 11 730
Nombre de chaines créées : 3 307

Temps d'exécution 8.2 secondes CPU



Traitement complet

Seuil du Bruit = 15

Nombre de chaines fusionnées = 1087
Nombre de chaines éliminées = 2090
Nombre de chaines restantes = 130

Temps d'exécution 13.04 secondes CPU

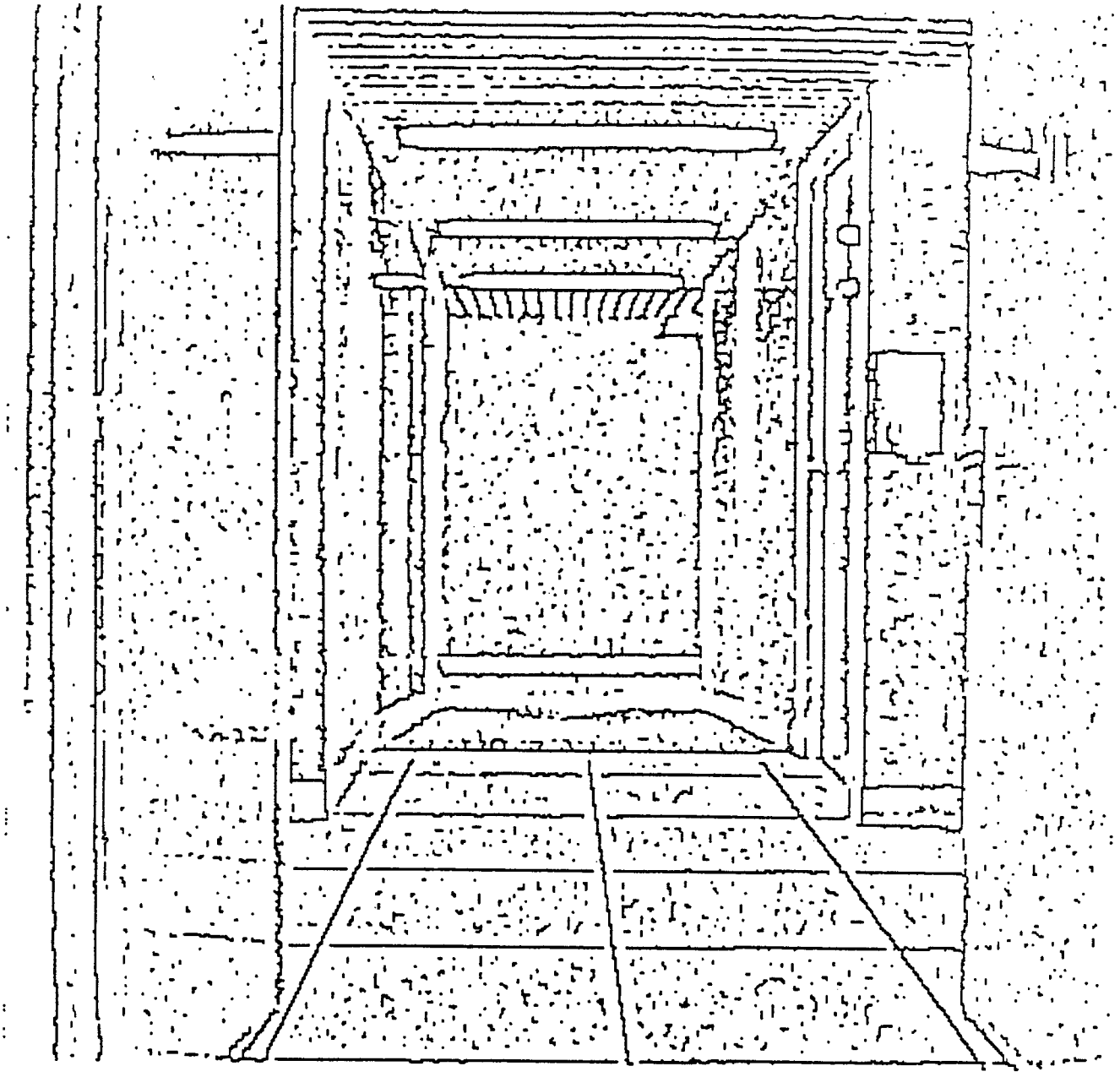
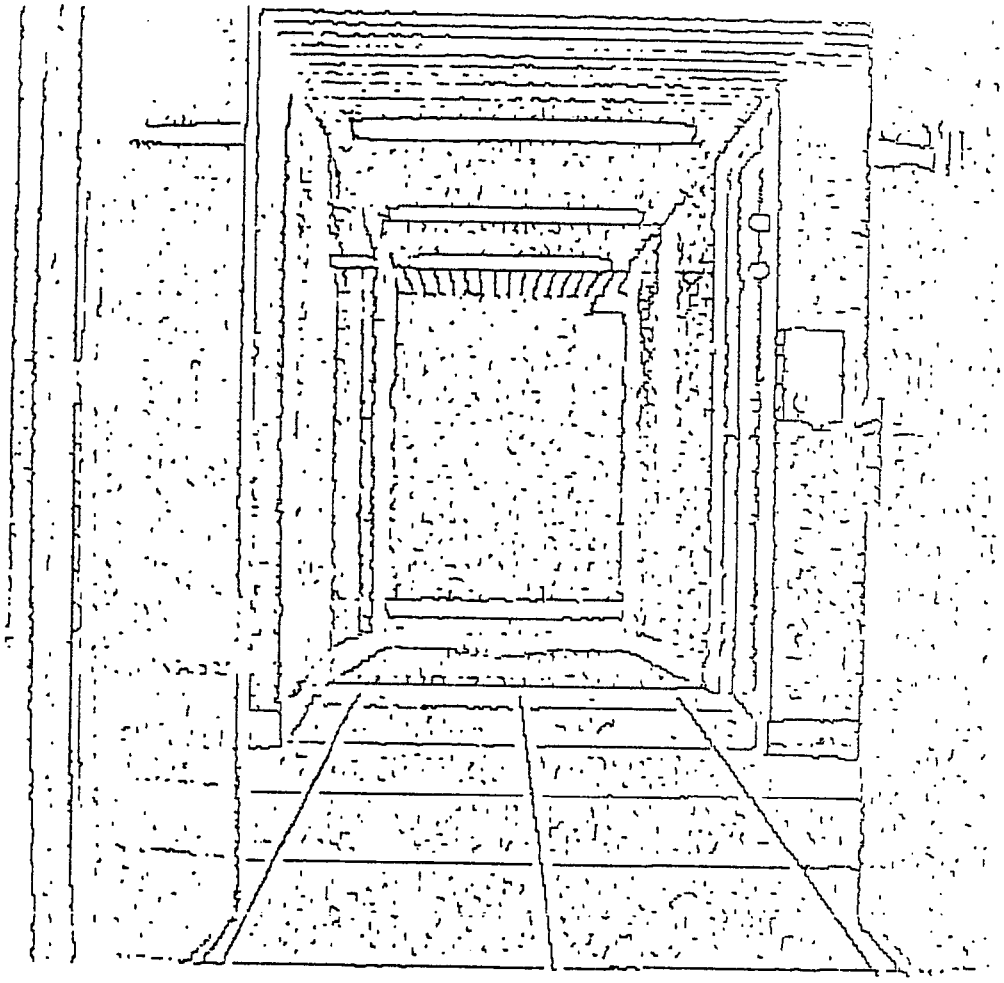


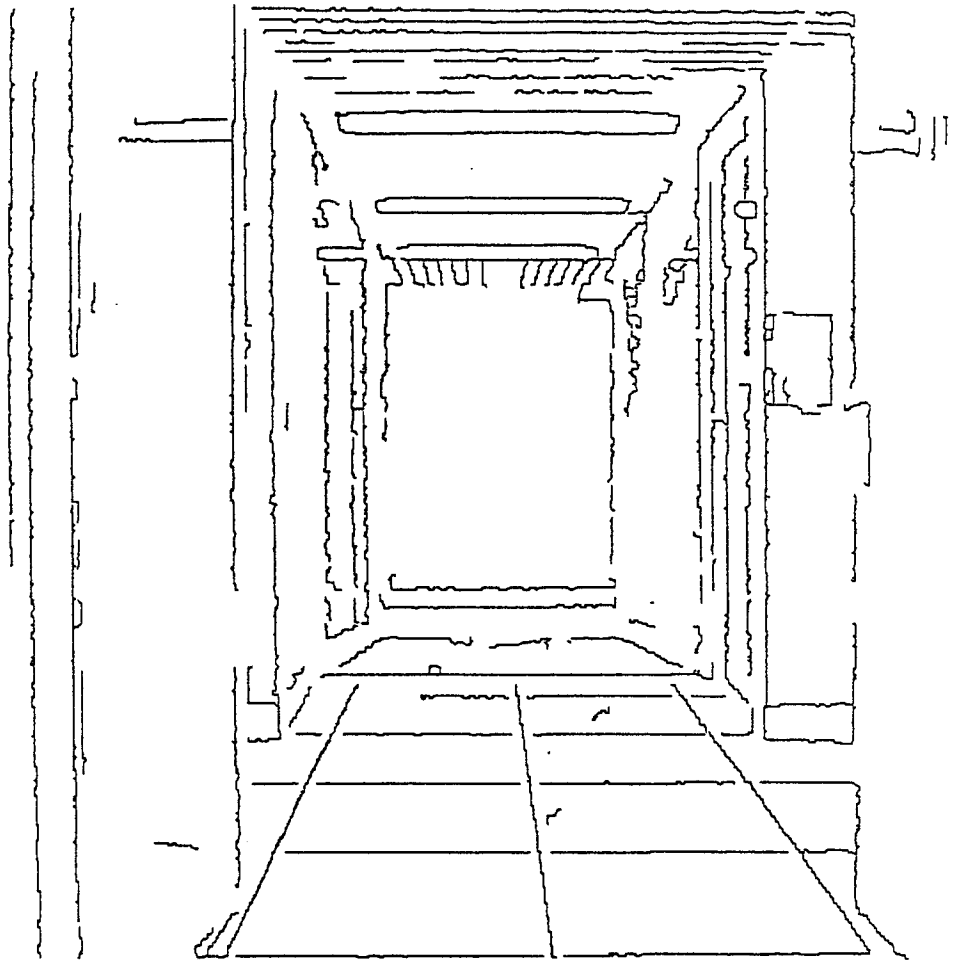
Image Brute : 512x512



Traitement : chainage seul

Nombre de pixels traités = 20 855
Nombre de chaines créées = 4 263

Temps d'exécution 22 secondes CPU



Traitement complet

Seuil du Bruit = 15

Nombre de chaines fusionnées = 1071
Nombre de chaines éliminées = 2900
Nombre de chaines restantes = 292

Temps d'exécution 31.48 secondes CPU

EXEMPLES DE TEMPS CPU

- IMAGE 256 x 256

Nombre de Pixels	Nombre de chaines initiales	Nombre de chaines finales	Temps de création des chaines(sec)	Temps Total (sec)	Temps moyen/pixel (sec)
0	0	0	3 7	3 7	-
1 390	68	15	45	45	$5\,7\,10^{-4}$
1 470	159	32	5	5 1	$8.8.10^{-4}$
2 305	286	63	5.3	5 6	$6.9.10^{-4}$
4 266	1 260	171	6	6 7	$5.4.10^{-4}$
4 518	698	153	6	6 5	$5.1.10^{-4}$
4 667	715	167	6 1	7.1	$5\,1.10^{-4}$
4 869	968	69	6.2	7 2	$5.1.10^{-4}$
8 536	1 365	266	7 4	10 6	$4.3.10^{-4}$
10 542	1 706	366	8.6	12 4	$4.6.10^{-4}$
11 730	3 362	250	8.6	12.2	$4.1.10^{-4}$

EXEMPLES DE TEMPS CPU

- IMAGE 512 x 512

Nombre de Pixels	Nombre de chaines initiales	Nombre de chaines finales	Temps de creation des chaines(sec)	Temps Total (sec)	Temps moyen/pixe (sec)
0	0	0	15.72	15.72	-
8 278	880	204	19.02	20.52	$4.0\ 10^{-4}$
20 855	4 263	292	22.05	31.48	$3.1.10^{-4}$

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique